Introduction to Python Programming (BPLCK205B)

Dr. Thyagaraju G S

Modules

- **Module1: Python Basics, Flow control, Functions**
- **Module2 : Lists, Tuples and Dictionaries**
- **Module3: Strings, Reading and Writing Files**
- **Module 4: Organizing Files and Debugging**
- **Module 5: Classes and Objects, Classes and Methods, Classes and Functions**

Module 1:

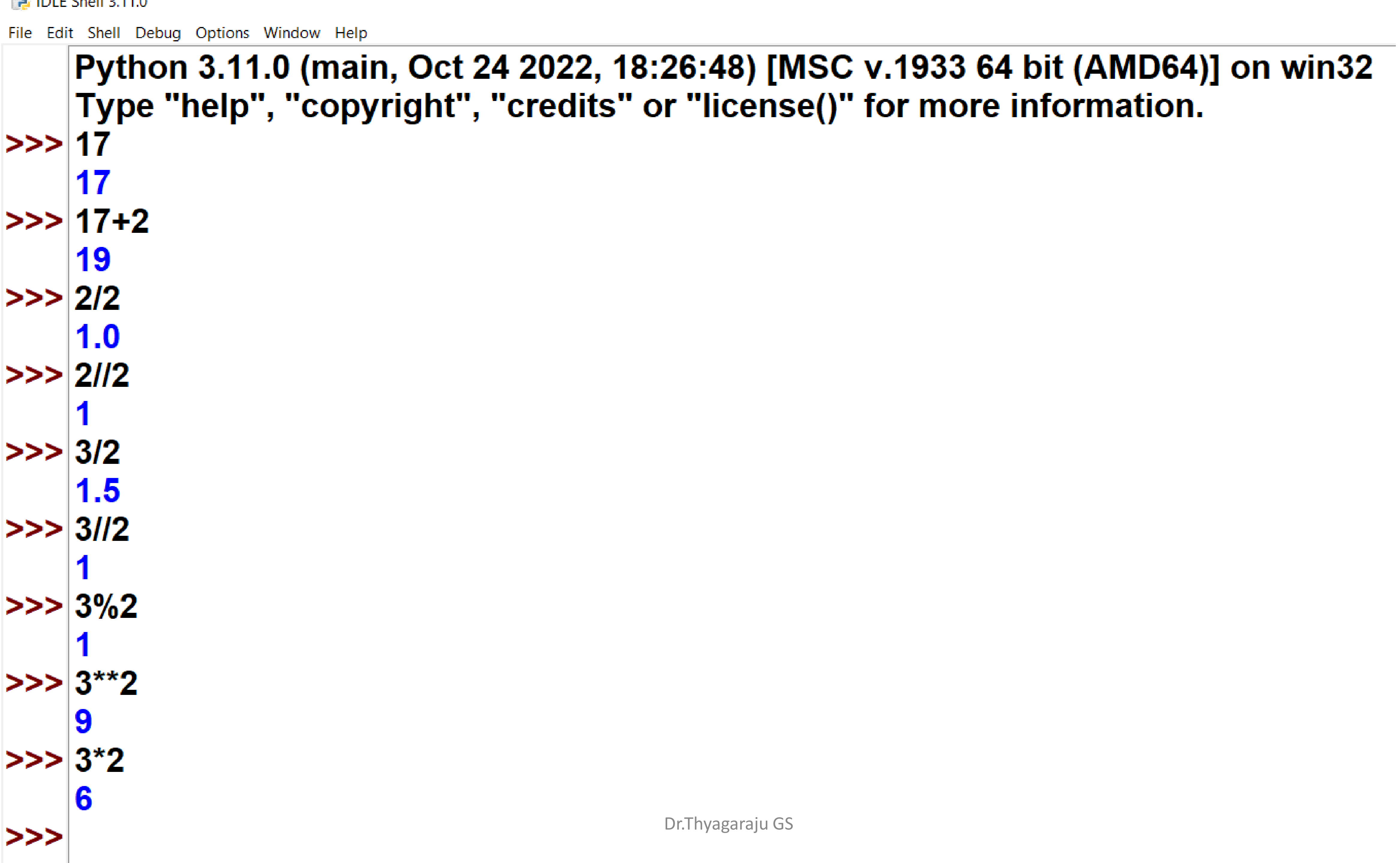
- **Python Basics**: Entering Expressions into the Interactive Shell, The Integer, Floating-Point, and String Data Types, String Concatenation and Replication, Storing Values in Variables, Your First Program, Dissecting Your Program,
- **Flow control:** Boolean Values, Comparison Operators, Boolean Operators, Mixing Boolean and Comparison Operators, Elements of Flow Control, Program Execution, Flow Control Statements, Importing Modules, Ending a Program Early with sys.exit(),
- **Functions:** def Statements with Parameters, Return Values and return Statements,The None Value, Keyword Arguments and print(), Local and Global Scope, The global Statement, Exception Handling, A Short Program: Guess the Number

1.1 Python Basics

- **Entering Expressions into the Interactive Shell,**
- **The Integer, Floating-Point, and String Data Types,**
- **String Concatenation and Replication,**
- **Storing Values in Variables,**
- **Your First Program, Dissecting Your Program,**

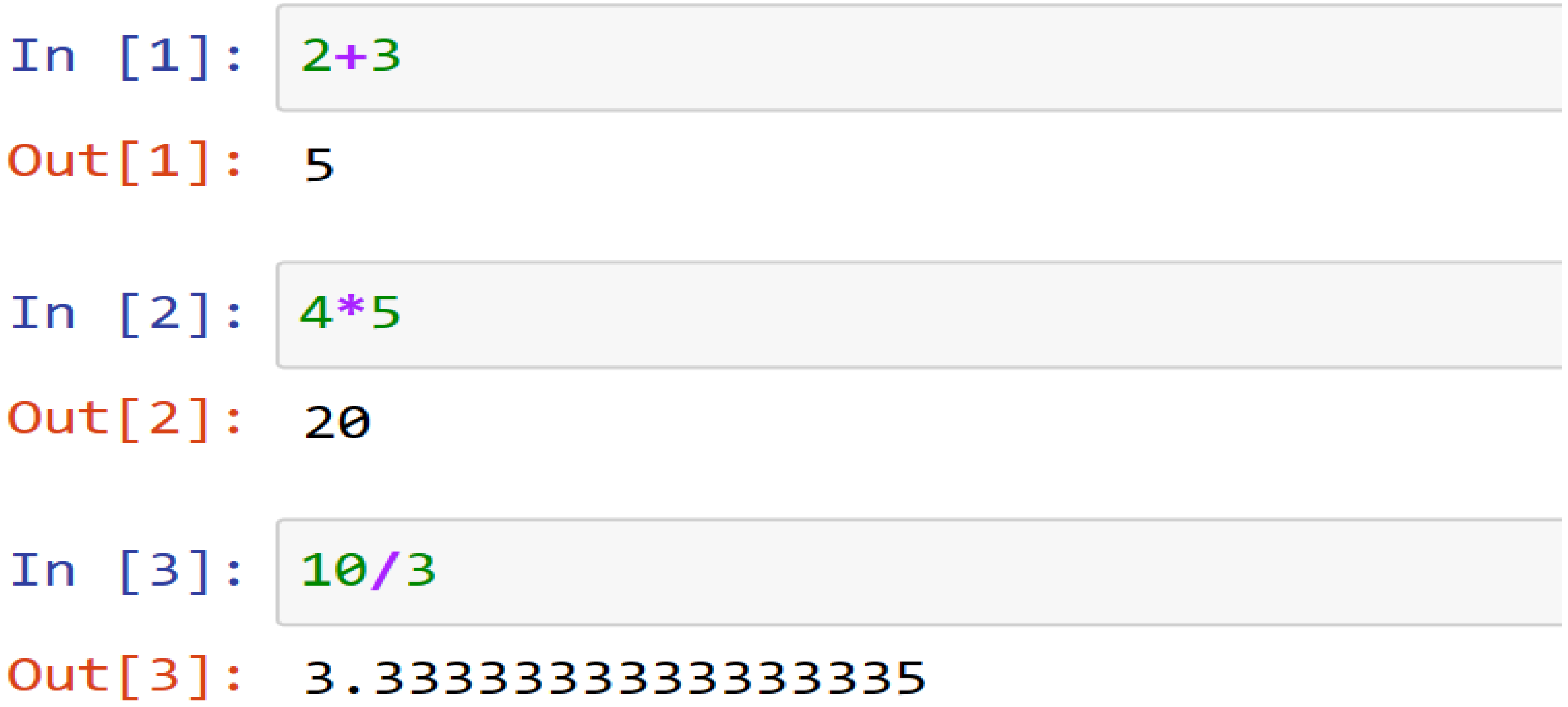
Entering Expressions into the Interactive Shell

- In Python, expressions are combinations of values, variables, operators, and function calls that can be evaluated to produce a result. They represent computations and return a value when executed. Here are some examples of expressions in Python:
- **Examples:** 17, x, x+17 , 1+2*2 , X**2, x**2 + y**2



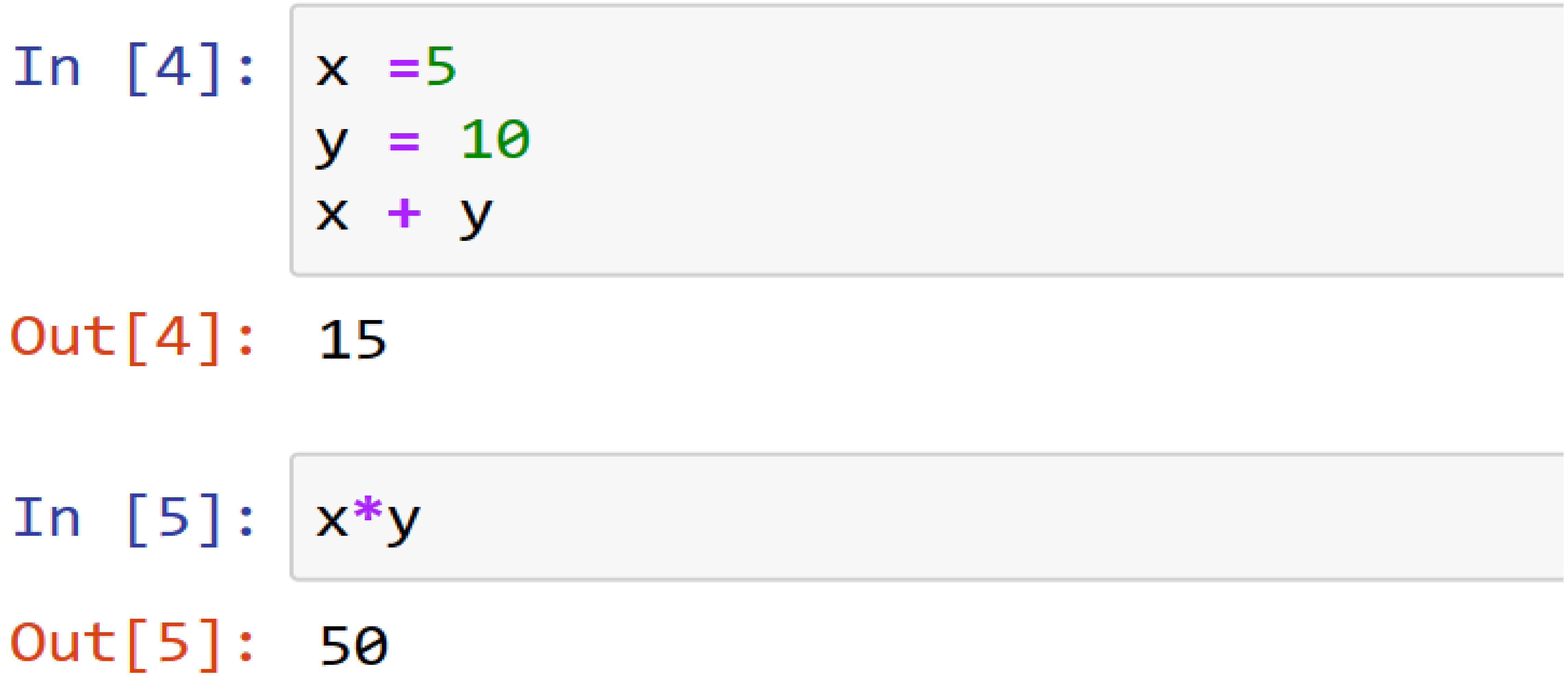
```
Python 3.11.0 (main, Oct 24 2022, 18:26:48) [MSC v.1933 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> 17
17
>>> 17+2
19
>>> 2/2
1.0
>>> 2//2
1
>>> 3/2
1.5
>>> 3//2
1
>>> 3%2
1
>>> 3**2
9
>>> 3*2
6
>>>
```

Arithmetic Operations



```
In [1]: 2+3
Out[1]: 5

In [2]: 4*5
Out[2]: 20

In [3]: 10/3
Out[3]: 3.3333333333333335
```

Variable Assignments



```
In [4]: x =5
        y = 10
        x + y

Out[4]: 15
```

```
In [5]: x*y

Out[5]: 50
```

Function Calls

```
abs(-10)
```

```
10
```

```
len("Hello World")
```

```
11
```

```
def add(a,b):
    return a + b
```

```
add(10,20)
```

```
30
```

Boolean Expressions

```
True and False
```

```
False
```

```
False and False
```

```
False
```

```
not True
```

```
False
```

Value

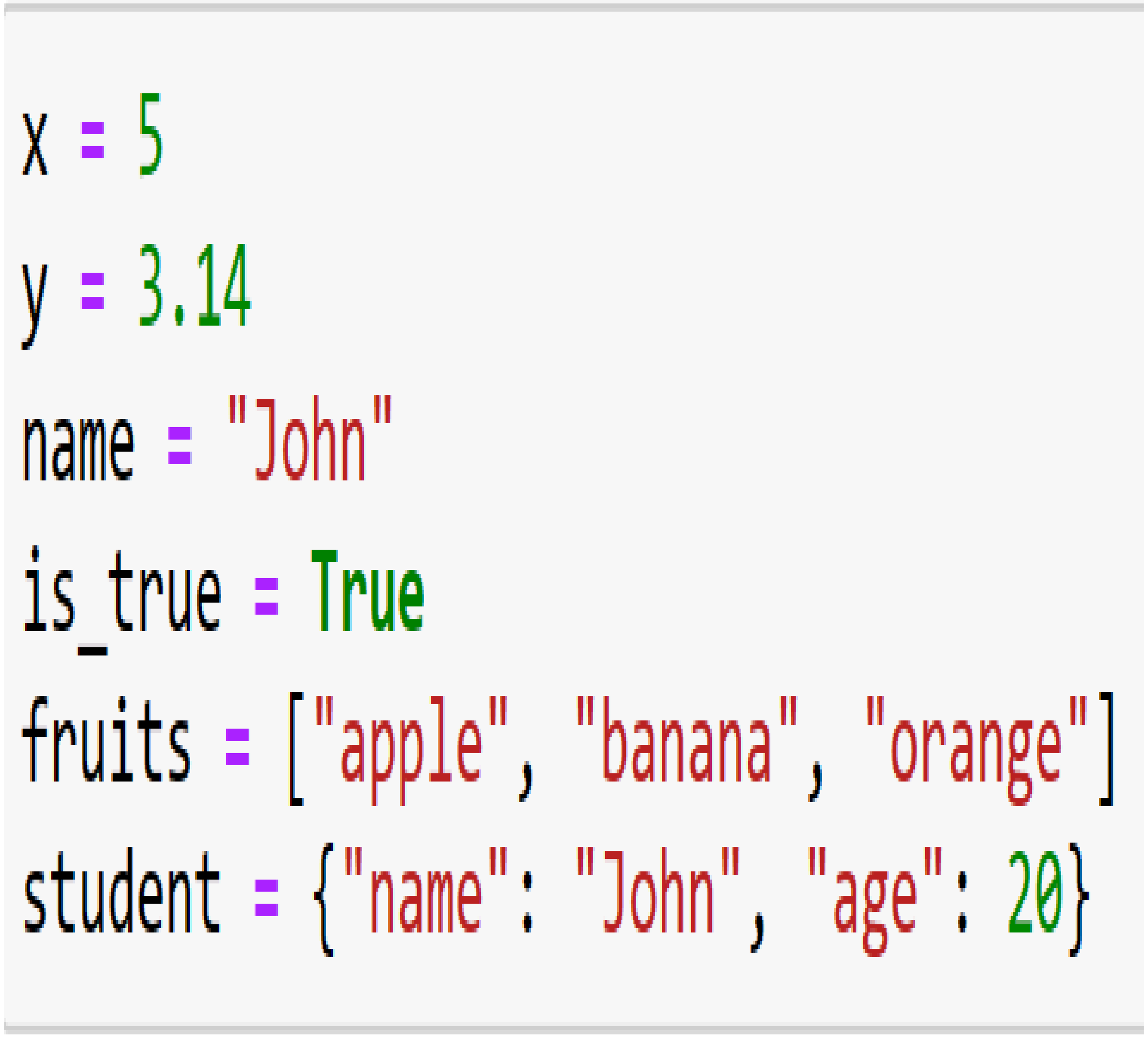
- **A value is a letter or a number.**
- **In Python, a value is a fundamental piece of data that can be assigned to variables, used in expressions, and manipulated by operations.**
- **Values can be of different types, such as numbers, strings, booleans, lists, tuples, dictionaries, and more. Each type of value has its own characteristics and behaviors.**

Examples

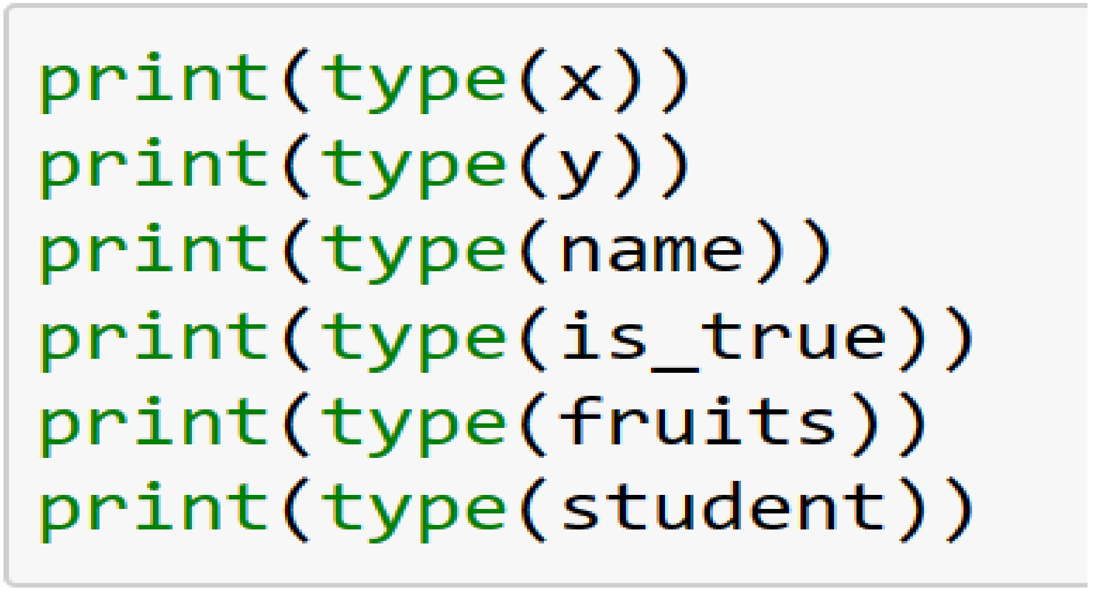
- **x = 10 # integer**
- **y = 3.14 # floating-point number**
- **z = 2 + 3j # complex number**
- **name = "John" # string**
- **message = 'Hello, World!'   # string**
- **is_true = True # Boolean Value**

type() function

- In Python, the type() function is used to determine the type of a given object or value.
- It returns the data type of the object as a result.



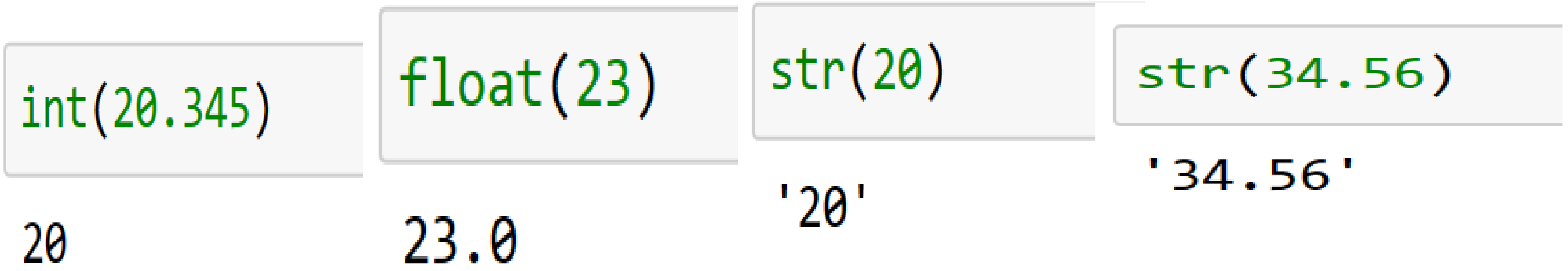
```python
x = 5
y = 3.14
name = "John"
is_true = True
fruits = ["apple", "banana", "orange"]
student = {"name": "John", "age": 20}
```



```python
print(type(x))
print(type(y))
print(type(name))
print(type(is_true))
print(type(fruits))
print(type(student))
```

```
<class 'int'>
<class 'float'>
<class 'str'>
<class 'bool'>
<class 'list'>
<class 'dict'>
```

int() ,str() and float() functions



```
int(20.345)
```

```
20
```

```
float(23)
```

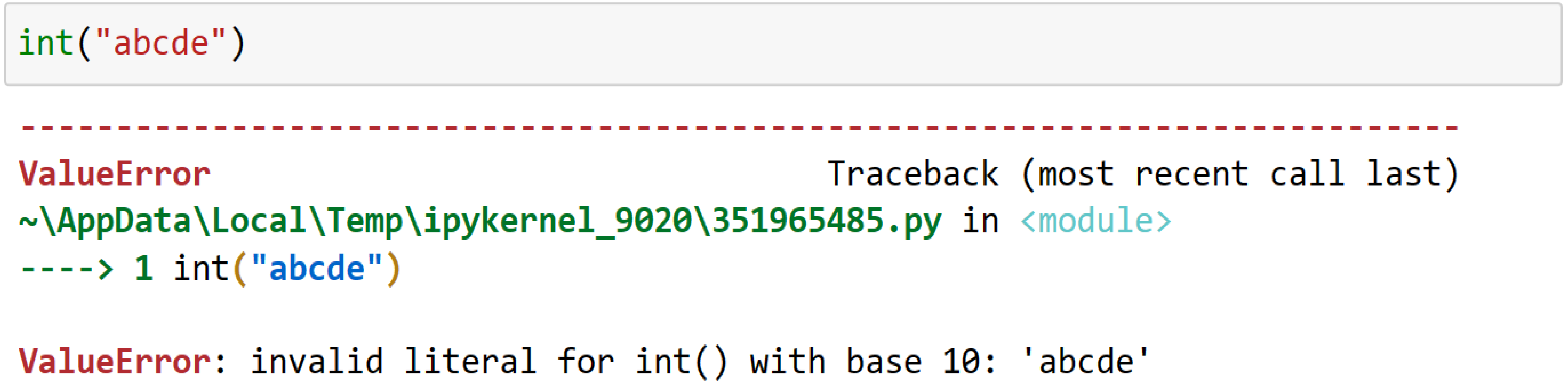
```
23.0
```

```
str(20)
```

```
'20'
```

```
str(34.56)
```

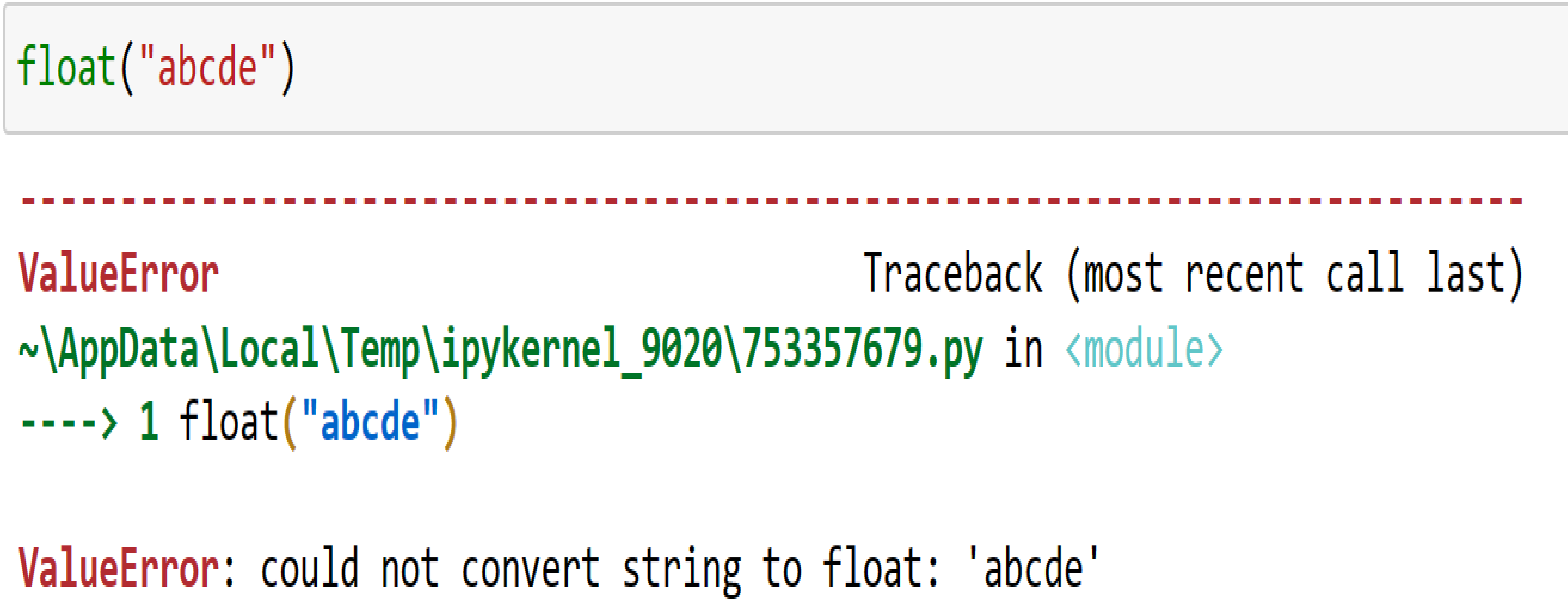
```
'34.56'
```



```
int("abcde")
```

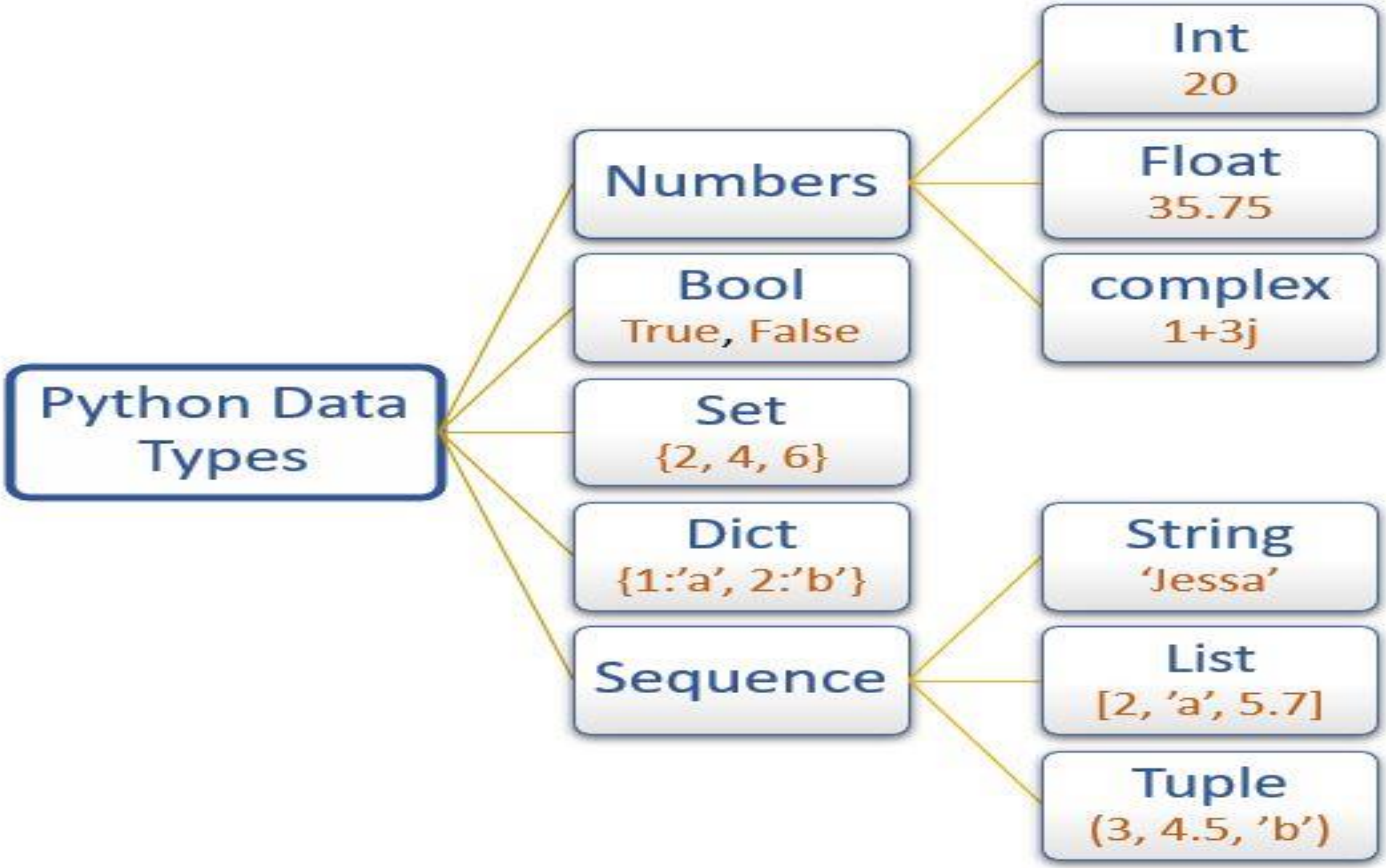
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_9020\351965485.py in <module>
----> 1 int("abcde")

ValueError: invalid literal for int() with base 10: 'abcde'
```

int() ,str() and float() functions



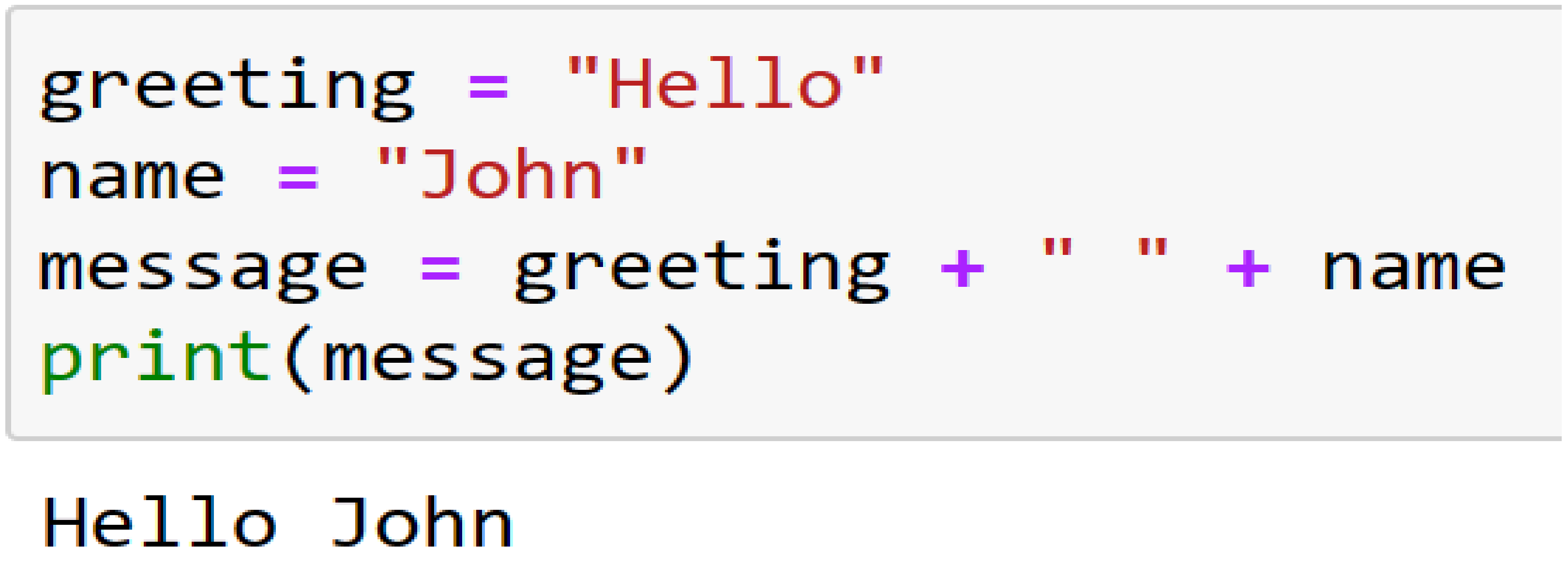
```python
float("abcde")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_9020\753357679.py in <module>
----> 1 float("abcde")

ValueError: could not convert string to float: 'abcde'
```


Python Data Types
Numbers
Int
20
Float
35.75
complex
1+3j
Bool
True, False
Set
{2, 4, 6}
Dict
{1:'a', 2:'b'}
Sequence
String
'Jessa'
List
[2, 'a', 5.7]
Tuple
(3, 4.5, 'b')

String Concatenation

- String concatenation is the process of combining two or more strings together to create a single string. In Python, you can concatenate strings using the **+** operator.
- Here's an example:

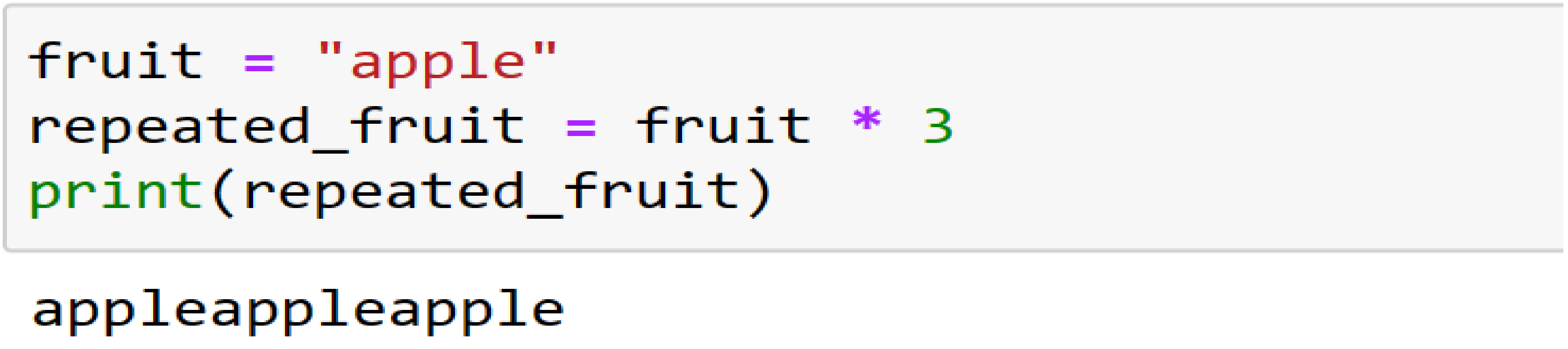


```python
greeting = "Hello"
name = "John"
message = greeting + " " + name
print(message)
```

```
Hello John
```

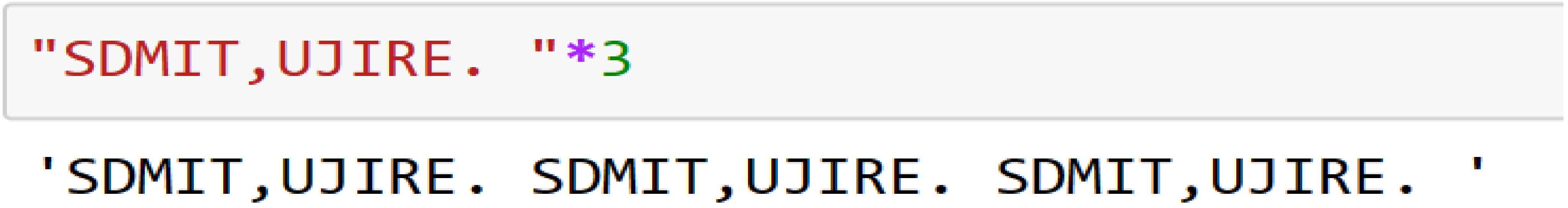
String Replication

- String replication allows you to repeat a string multiple times. In Python, you can replicate a string by using the * operator.
- Here's an example:



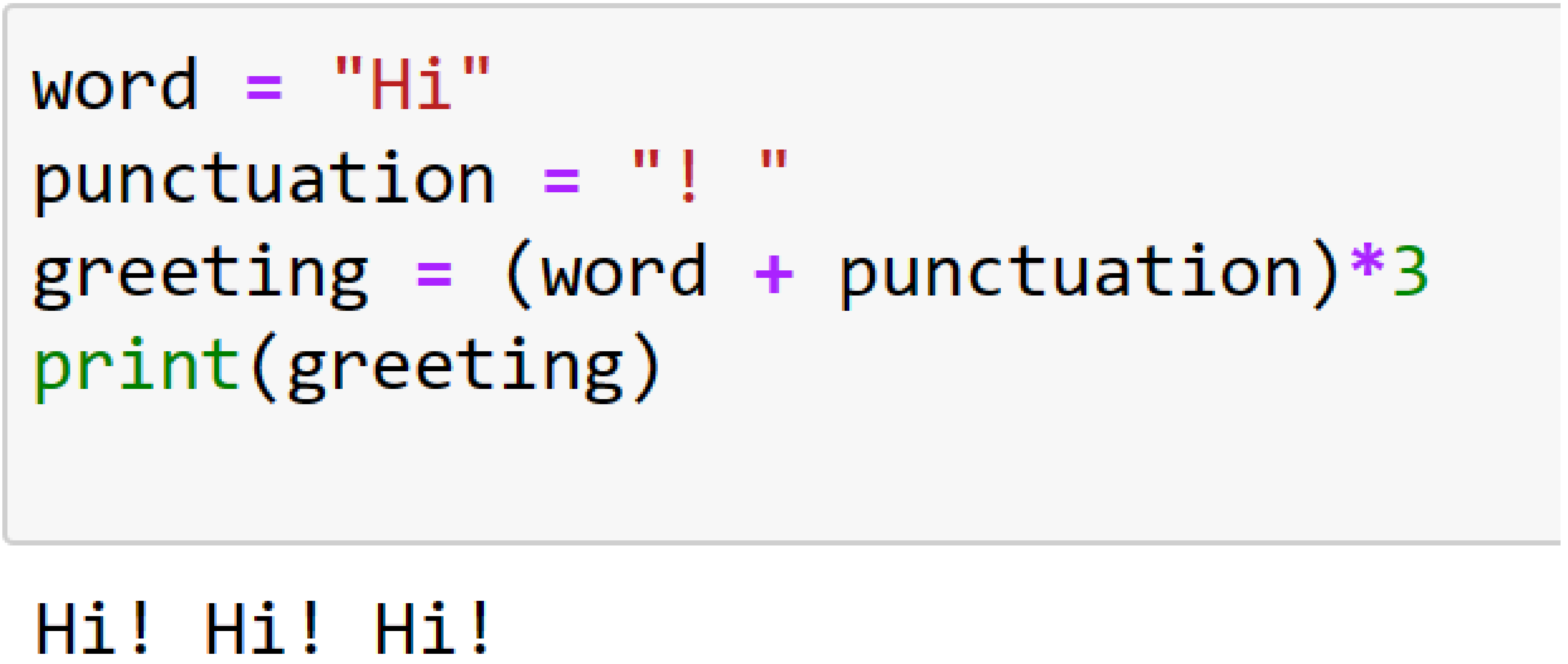
```
fruit = "apple"
repeated_fruit = fruit * 3
print(repeated_fruit)
```

```
appleappleapple
```



```
"SDMIT,UJIRE. "*3
```

```
'SDMIT,UJIRE. SDMIT,UJIRE. SDMIT,UJIRE. '
```



```python
word = "Hi"
punctuation = "! "
greeting = (word + punctuation)*3
print(greeting)
```

```
Hi! Hi! Hi!
```

Variable

- A variable is a name that refers to a value. In Python, a variable is a named storage location that holds a value.
- An assignment statement creates new variables as illustrated in the example below:

  x = 10

Examples

Message = ‘*Python Programming* ‘,

p =1000, **t**= 2**, r**=3.142,

Si = p*t*r/100,

pi = 3.141592653589793 1,

area_of _circle = *pi*r*r.*

To know the type of the variable one can use type () function.

Ex: type(p)

To display the value of a variable, you can use a print statement:

Ex: print (Si) ; print(pi)

Rules for writing Variable names

1. **Variable names** can be a combination of letters in lowercase (a to z) or uppercase (A to Z) or digits (0 to 9) or an underscore (_).
2. **Variable names** cannot start with a number/digit.
3. **Keywords** cannot be used as Variable names.
4. **Special symbols** like **!, @, #, $, %** etc. cannot be used in Variable names.
5. **Variable names** can be of any length.
6. **Variable name** must be of single word.

Valid Variable Names and **Invalid** Variable Names

Valid Variable Names	Invalid Variable Names
python12	current- account(hyphens are not allowed)
Simple	savings account (spaces are not allowed)
interest_year	4freinds (can’t begin with a number)
_rate_of_interest	1975 (can’t begin with a number)
_spam	10April_$ (cannot begin with a number and special characters like $ are not allowed)
HAM	Principle#@(special characters like # and @ are not allowed)
account1234	‘bear’ (special characters like ‘ is not allowed)

Storing Values in a Variables

- **Values** can be stored in a variable using an **Assignment statement**.
- **An assignment** statement consists of a variable name, an equal (=) sign and the value to be stored.

Example 1:

```
x = 40
```

Example 2:

```
a, b, c = 1, 2, 3
```

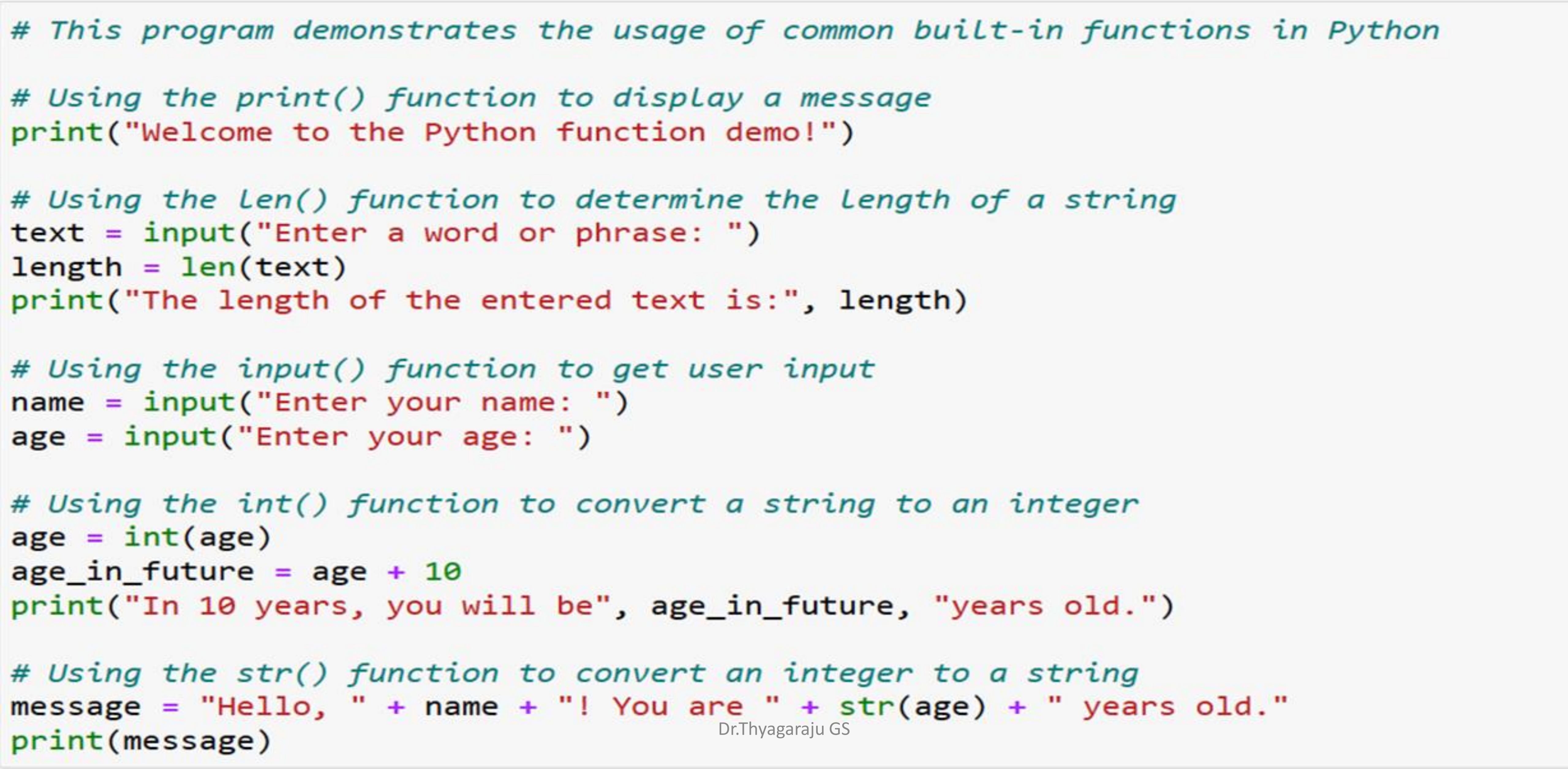
Example 3:

```
x = 5
y = 3
result = x + y
```

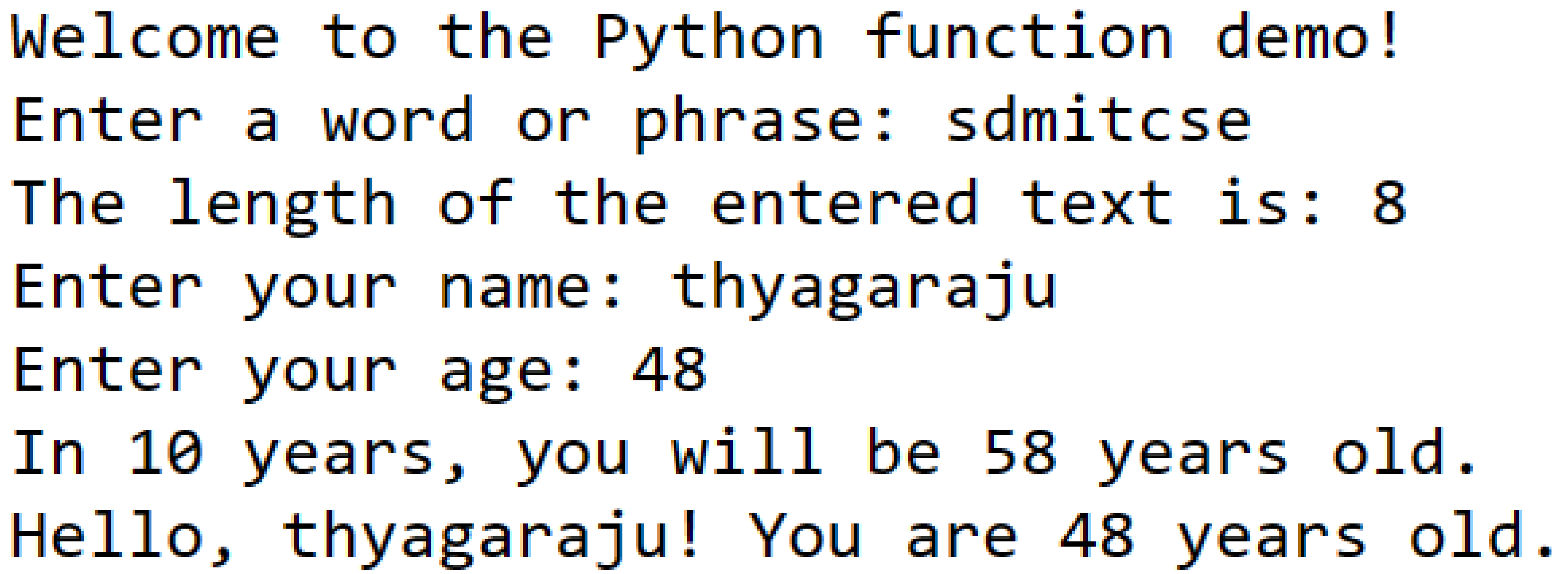
Example 4:

```
x = 10
x = x + 5  # x is updated to 15
```

Dissecting the Simple Program



```python
# This program demonstrates the usage of common built-in functions in Python

# Using the print() function to display a message
print("Welcome to the Python function demo!")

# Using the len() function to determine the length of a string
text = input("Enter a word or phrase: ")
length = len(text)
print("The length of the entered text is:", length)

# Using the input() function to get user input
name = input("Enter your name: ")
age = input("Enter your age: ")

# Using the int() function to convert a string to an integer
age = int(age)
age_in_future = age + 10
print("In 10 years, you will be", age_in_future, "years old.")

# Using the str() function to convert an integer to a string
message = "Hello, " + name + "! You are " + str(age) + " years old."
print(message)
```



```
Welcome to the Python function demo!
Enter a word or phrase: sdmitcse
The length of the entered text is: 8
Enter your name: thyagaraju
Enter your age: 48
In 10 years, you will be 58 years old.
Hello, thyagaraju! You are 48 years old.
```

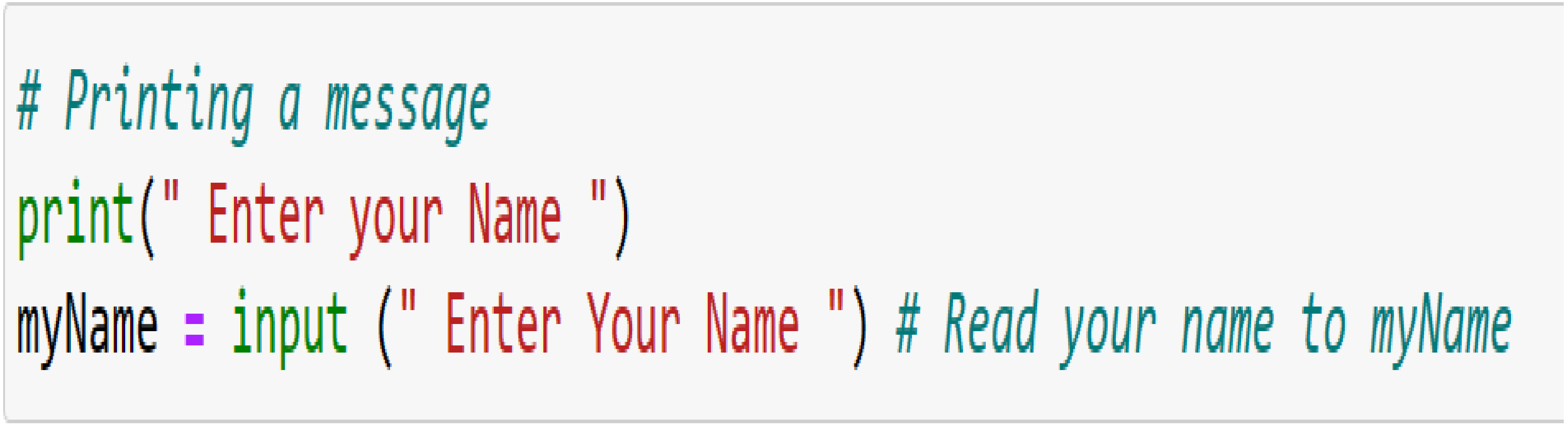
Comments

- Comments are readable explanation or descriptions that help programmers better understand the intent and functionality of the source code.
- Comments are completely ignored by interpreter.

Advantages of Using Comments:

1. Makes code more readable and understandable.
2. Helps to remember why certain blocks of code were written.
3. Can also be used to ignore some code while testing other blocks of code.

Single Line Comments in Python:



```python
# Printing a message
print(" Enter your Name ")
myName = input (" Enter Your Name ") # Read your name to myName
```

Multiline Comments

1. Using # at the beginning of each line of comment on multiple lines

Example:

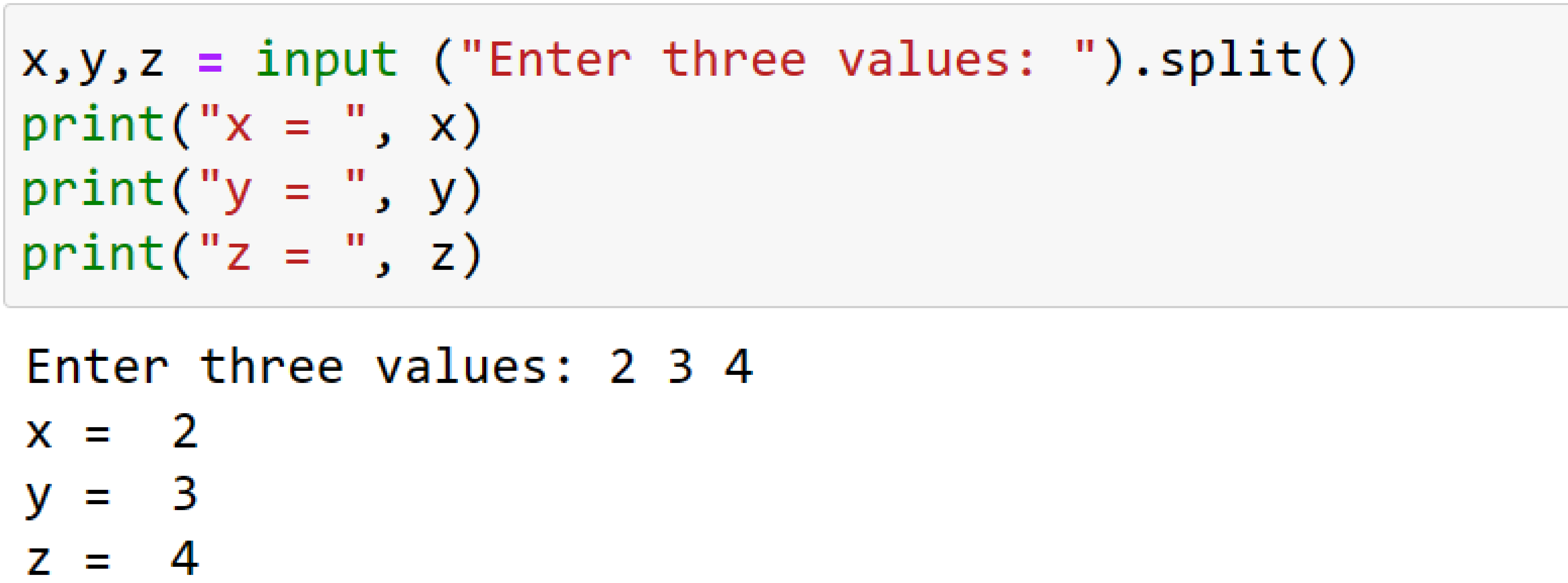
```
# It is a
# multiline
# comment
```

2. **Using String Literals ''' at the beginning and end of multiple lines**

Example:

```
'''
I am a
Multiline comment!
'''
```

Reading multiple values using input()

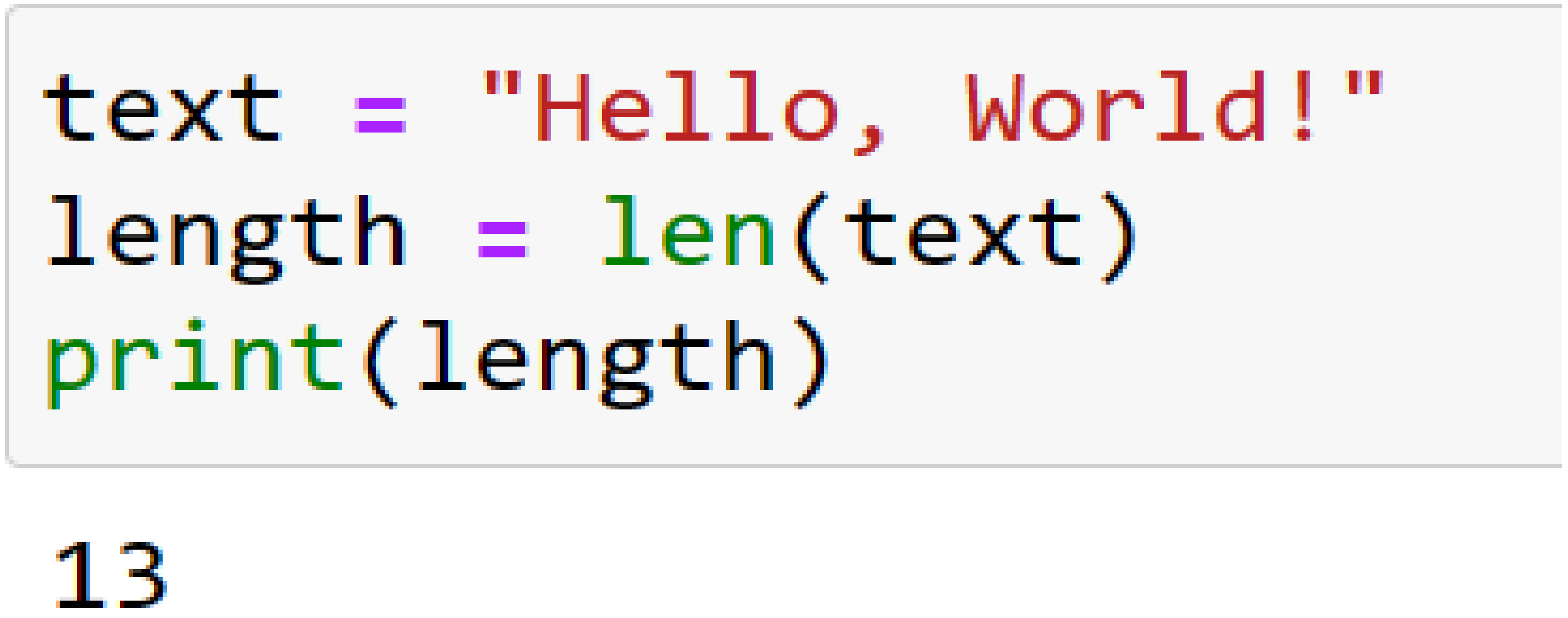


```python
x,y,z = input ("Enter three values: ").split()
print("x = ", x)
print("y = ", y)
print("z = ", z)
```

```
Enter three values: 2 3 4
x =  2
y =  3
z =  4
```

The len() Function

- In Python, the **len()** function is used to determine the length of an object, such as a **string, list, tuple, or any other iterable**.



```
text = "Hello, World!"
length = len(text)
print(length)
```

```
13
```

Operators and operands

- Operators are special symbols that represent computations like addition and multiplication. The values the operator is applied to are called operands.
- The operators **+, -, *, /, and **** perform *addition, subtraction, multiplication, division, and exponentiation*, as in the following examples:

Operator	Operation	Example	Evaluates to
**	Exponent	5**3	125
%	Modulus/Remainder	33%7	5
//	Integer Division/Floored quotient	33//5	6
/	Division	23/7	3.2857142857142856
*	Multiplication	7*8	56
-	Subtraction	8 – 5	3
+	Addition	7+ 3	10

Order of operations

- When more than one operator appears in an expression, the order of evaluation depends on the rules of precedence.
- **PEMDAS** order of operation is followed in Python:
  - **Parentheses** have the highest precedence and can be used to force an expression to evaluate in the order you want.
  - **Exponentiation** has the next highest precedence,
  - **Multiplication and Division** have the same precedence, which is higher than
  - **Addition and Subtraction**, which also have the same precedence.
  - **Operators with the same precedence** are evaluated from left to right.
-

Example1

$$\begin{tikzcd}
(5 - 1) * ((7 + 1) / (3 - 1)) \arrow[d] \\
4 * ((7 + 1) / (3 - 1)) \arrow[d] \\
4 * (\ 8\ ) / (3 - 1)) \arrow[d] \\
4 * (\ 8\ ) / (\ 2\ ) \arrow[d] \\
4 * 4.0 \arrow[d] \\
16.0
\end{tikzcd}$$

Example2

(5-2)*((8+4)/(5-2))

3 * ((8+4)/(5-2))

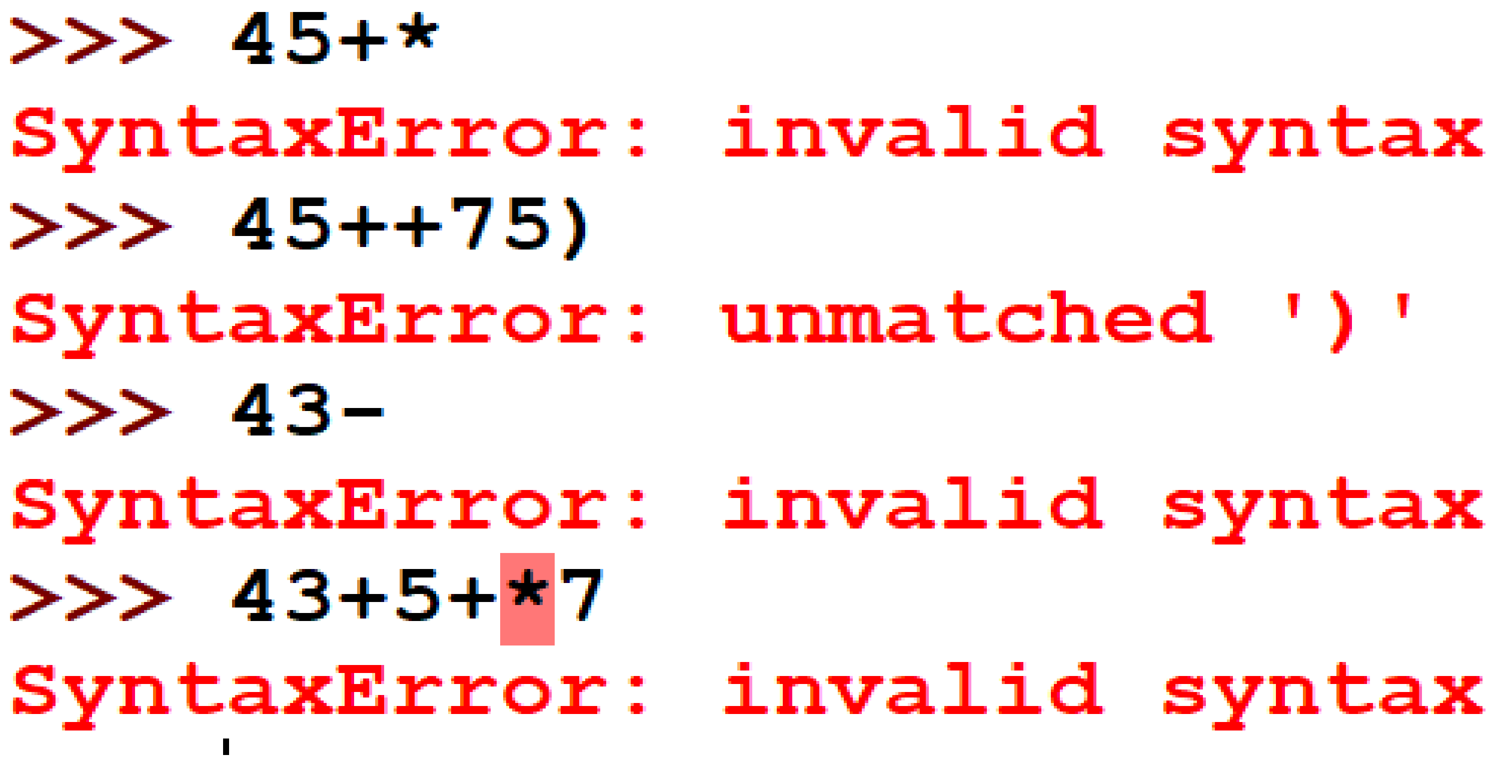
3*(12/(5-2))

3*(12/3)

3*4.0

12.0

Example3 : Invalid Expressions



```
>>> 45+*
SyntaxError: invalid syntax
>>> 45++75)
SyntaxError: unmatched ')'
>>> 43-
SyntaxError: invalid syntax
>>> 43+5+*7
SyntaxError: invalid syntax
```

Python Character Set :

- The set of valid characters recognized by Python like letter, digit or any other symbol. The latest version of Python recognizes Unicode character set. Python supports the following character set:

- **Letters** : A-Z ,a-z
- **Digits** :0-9
- **Special Symbols** : space +-/***()[]{}//=!= == <>,"""",;: %!#?$& ^⇔=@_
- **White Spaces** : Blank Space, tabs(->), Carriage return , new line , form feed
- **Other Characters** : All other 256 ACII and Unicode characters

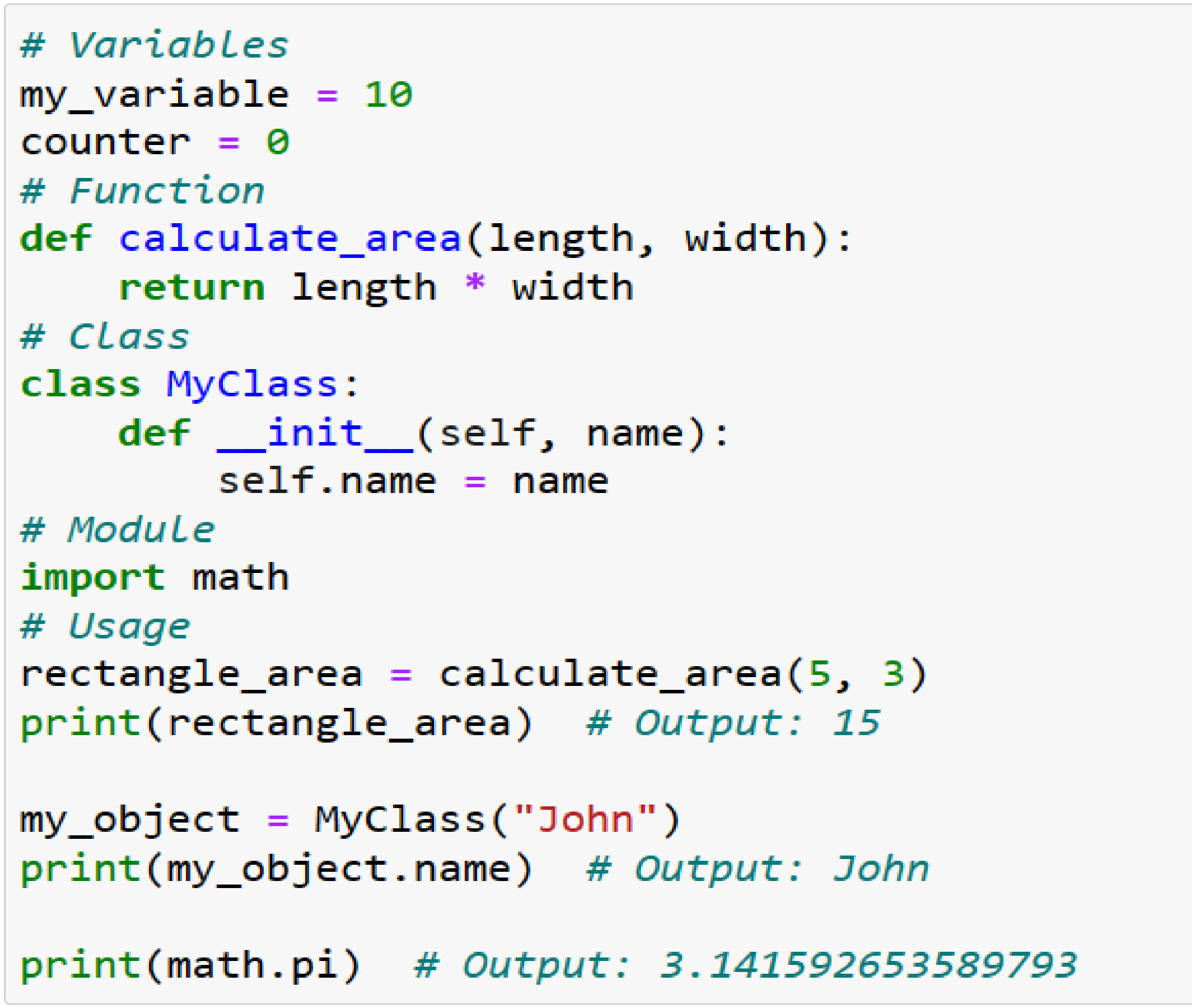
Python Tokens:

A **token (lexical unit)** is the smallest element of Python script that is meaningful to the interpreter. Python has following categories of tokens:

1. Identifiers
2. Literals
3. Operators
4. Delimiters
5. Keywords

1.Identifiers

- **Identifiers** are names that you give to a variable , class or Function.
- There are certain rules for naming identifiers similar to the variable declaration rules , such as :
  - No Special character except_ ,
  - Keywords are not used as identifiers ,
  - the first character of an identifier should be _ underscore or a character ,
  - but a number is not valid for identifiers and
  - identifiers are case sensitive



```python
# Variables
my_variable = 10
counter = 0
# Function
def calculate_area(length, width):
    return length * width
# Class
class MyClass:
    def __init__(self, name):
        self.name = name
# Module
import math
# Usage
rectangle_area = calculate_area(5, 3)
print(rectangle_area)  # Output: 15

my_object = MyClass("John")
print(my_object.name)  # Output: John

print(math.pi)  # Output: 3.141592653589793
```

In the above example, we have used identifiers like **my_variable, counter, calculate_area, MyClass, and math.**

2. Literals in Python

- In Python, literals are the raw, literal values that are used to represent data in the code. They are fixed values that are directly assigned to variables or used as constants.
- Python supports various types of literals, including numeric literals, string literals, Boolean literals, and more.

1. Numeric literals: Numeric literals represent numeric values such as integers, floating-point numbers, and complex numbers.
Examples: x = 10, y = 3.14, z = 2 + 3j

2. **String literals:** String literals represent sequences of characters enclosed in either single quotes (') or double quotes (").
Examples: name = 'John', sage = "Hello, world!“

3. **Boolean literals**: Boolean literals represent the truth values True and False.
Examples: is_valid = True

4. **None literal:** The None literal represents the absence of a value or a null value. It is often used to indicate the absence of a meaningful result or as an initial value for variables.
Example: result = None

4. **Operator Literals** : Operator literals include arithmetic operators, comparison operators, assignment operators, logical operators, and more.
Examples: +,-,/,//,%,*,**, <,>,!=,==,and,or,not,etc.

3.Operators

- A Symbol or a word that performs some kind of operation on given values and returns the result.
- There are 7 types of operators available for Python: *Arithmetic Operator, Assignment Operator, Comparison Operator, Logical Operator, Bitwise Operator, Identity Operator and Membership Operator*.

1. **Arithmetic operators:** +, -, *, /, %, **, //
2. **Assignment operators:** =, +=, -=, *=, /=, %=, **=, //=
3. **Comparison operators:** ==, !=, >, <, >=, <=
4. **Logical operators:** and, or, not
5. **Bitwise operators:** &, |, ^, ~, <<, >>
6. **Membership operators:** in, not in
7. **Identity operators:** is, is not

4. Delimiters

- Delimiters are the symbols which can be used as separators of values or to enclose some values.
- **Examples** : Comma (,),Colon (:),Parentheses ((and)),Square brackets ([and]),Curly braces ({ and }),Quotation marks (' and ") and Backslash (\)

5. Keywords

- The **reserved words** of Python which have a special fixed meaning for the interpreter are called keywords.
- No **keyword** can be used as an identifier or variable names. There are 36 keywords in python as listed below:

```python
import keyword
print(keyword.kwlist)
print("\nTotal Number of Keywords: ",len(keyword.kwlist))
```

```
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'aw
ait', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'fi
nally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonloca
l', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

Total Number of Keywords:  36
```

1.2 Flow Control

Syllabus:

- Boolean Values, Comparison Operators, Boolean Operators, Mixing Boolean and Comparison Operators,
- Elements of Flow Control, Program Execution, Flow Control Statements,
- Importing Modules, Ending a Program Early with **sys.exit().**

Boolean Values:

- A Boolean value is either true or false.
- In Python the two Boolean Values are **True** and **False** and the Python type is bool.

type(**True**)
# output : **bool**

type(**False**)
# output : **bool**

type(**true**)
# output: Name Error : name “ true” is not defined

type(**false**)   # output: Name Error : name “ false” is not defined

context = True
print(context) #output : **True**

Boolean Expressions

- A **Boolean expression** is an expression that evaluated to produce a result which is a Boolean value.

```
5 == (1+4)  # output : True

5 == 6          #  output: False
```

```
P = “hel”
P + “lo” == “hello”  # output: True
```

Comparison Operators

Operator	Meaning
==	Equal to
!=	Not Equal to
<	Less than
>	Greater than
<=	Les than or equal to
>=	Greater than or equal to

```
55 == 55 # output: True
55 == 79 # output: False
7!=10     # output : True
7!=7       #output : False
True == True # output: True
True != False # output: True
```

```
12< 13  # output: True
55.55 > 66.75  # output : False
```

```
"tag"< = 2
 # output :
Type error : '<' is not supported between instances of 'str' and 'int'.
```

Difference between == and = Operator

=	==
It is an assignment operator	It is a comparison operator
It is used for assigning the value to a variable	It is used for comparing two values. It returns 1 if both the value is equal otherwise returns 0
Constant term cannot be place on left hand side Example: 1= x; is invalid	Constant term can be placed in the left-hand side. Example: 1 ==1 is valid and return 1

Boolean Operators:

True and True # output : True

True and False # output : False

False and True # output : False

False and False # output :False

True or True # output : True

True or False # output : True

False or True # output : True

False or False # output :False

Op1	**Op2**	**Op1 and Op2**
True	**False**	**False**
False	**True**	**False**
False	**False**	**False**
True	**True**	**True**

Op1	Op2	Op1 or Op2
True	False	True
False	True	True
False	False	False
True	True	True

Not operator:

- It is a unary operator and evaluates the expression to opposite value true or false as illustrated below :

not True # output : False

not False # output : True

not not not not True # output : True

op	not op
True	False
False	True

Mixing Boolean and Comparison Operators

x = 10

y = 20

x<y and x>y **# output : False**

(x<y) and (x!=y) or (x*2) and (x<20 or y<20) **# output : True**

2+2 == 4 and not 2+2 == 5 and 2*2 == 2+2   **#output : True**

5*7 +8 == 7 or not 5+7 ==10 and 5*4 ==20   **#output : True**

Elements of Flow Control

Flow control statements often starts with condition followed by a block of code called clause. The two elements of Flow Control are discussed below:

a) Conditions:

Conditions are Boolean expressions with the boolean values True or False. Flow control statements decides what to do based on the condition whether it is true or false.

b) Blocks of Code /Clause:

The set of more than one statements grouped with same indentation so that they are syntactically equivalent to a single statement is known as Block or Compound Statement. One can tell when a block begins, and ends based on indentation of the statements. Following are three rules for blocks:

1. **Blocks begin when the indentation increases**
2. **Blocks can have nested blocks**
3. **Blocks end when the indentation decreases to zero**

Example 1:

```
x = int(input(“Enter a number:  “)) # Block
if  x>=10: # Condition
          x = x + 25    # Block belonging to if
          y = x
          print(x,y)
print(“Next statement”) # Next Block
```

Example 2: Nested Blocks

```
N = int(input(“Enter a number of your choice”))
if  n > 0:
        print(“Positive”) # Block  of outer if
        if n%2==0:
                print(“Multiple of Two”) # Block of Inner if
print("End”)
```

Flow Control statements

Flow control statements in Python are used to control the order of execution and make decisions based on certain conditions. The main flow control statements in Python include

Conditional Control Statements:

- **if statement**: Executes a block of code if a specified condition is true.
- **elif statement**: Allows you to check additional conditions if the previous if or elif conditions are false.
- **else statement**: Executes a block of code if none of the previous conditions are true.

Looping Statements:

- **for loop:** Iterates over a sequence (such as a list, tuple, string, or range) and executes a block of code for each item in the sequence.
- **while loop:** Repeats a block of code as long as a specified condition is true.

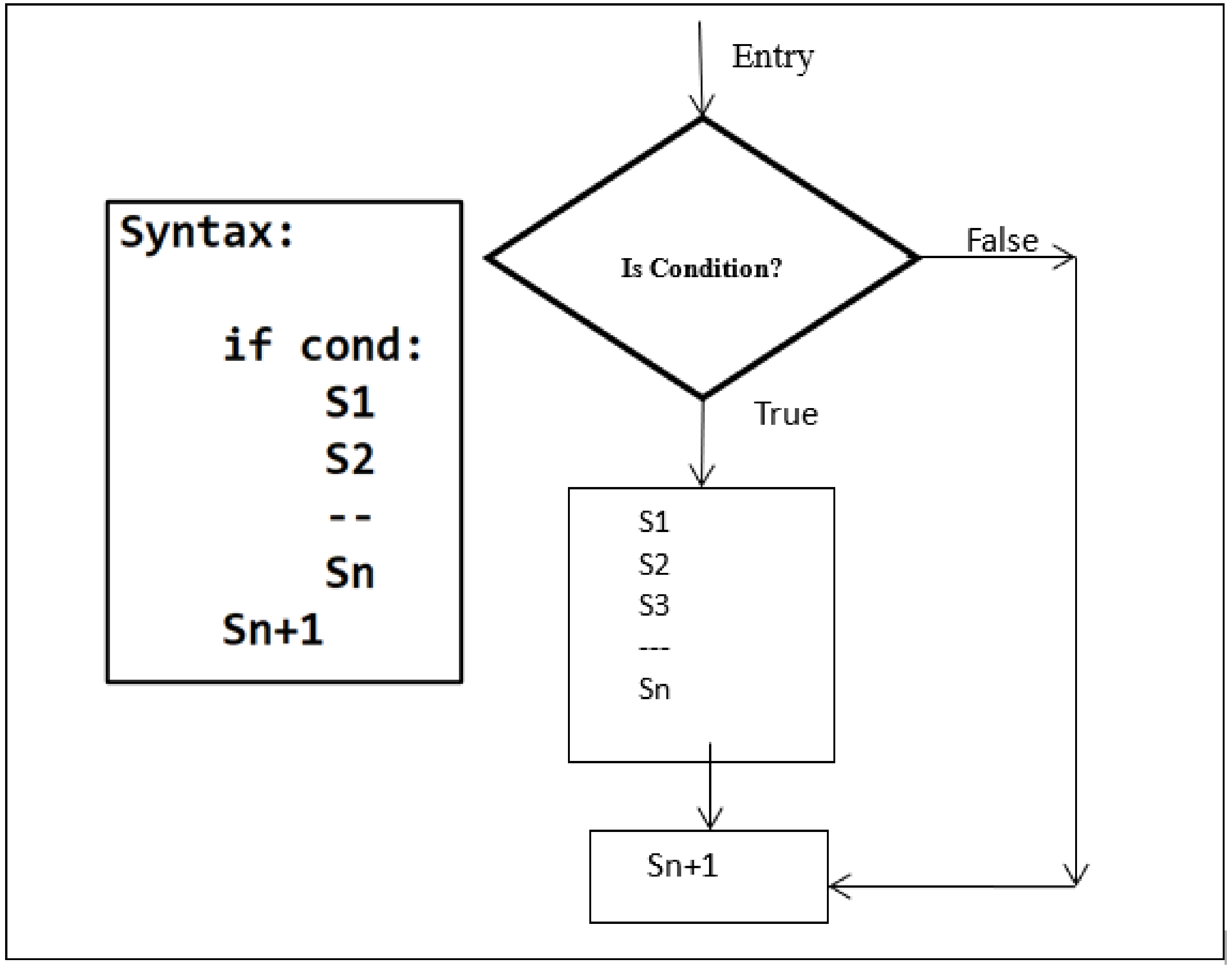
Loop Control Statements:

- **break statement:** Terminates the innermost loop and continues with the next statement after the loop.
- **continue statement**: Skips the rest of the current iteration and moves to the next iteration of the loop.
- **pass statement**: Acts as a placeholder, allowing you to create empty code blocks without causing syntax errors

Exception Handling Statements:

- **try statement:** Defines a block of code where exceptions might occur.
- **except statement**: Specifies the code to execute if a specific exception occurs within the try block.
- **finally statement**: Defines a block of code that will be executed regardless of whether an exception occurred or not.

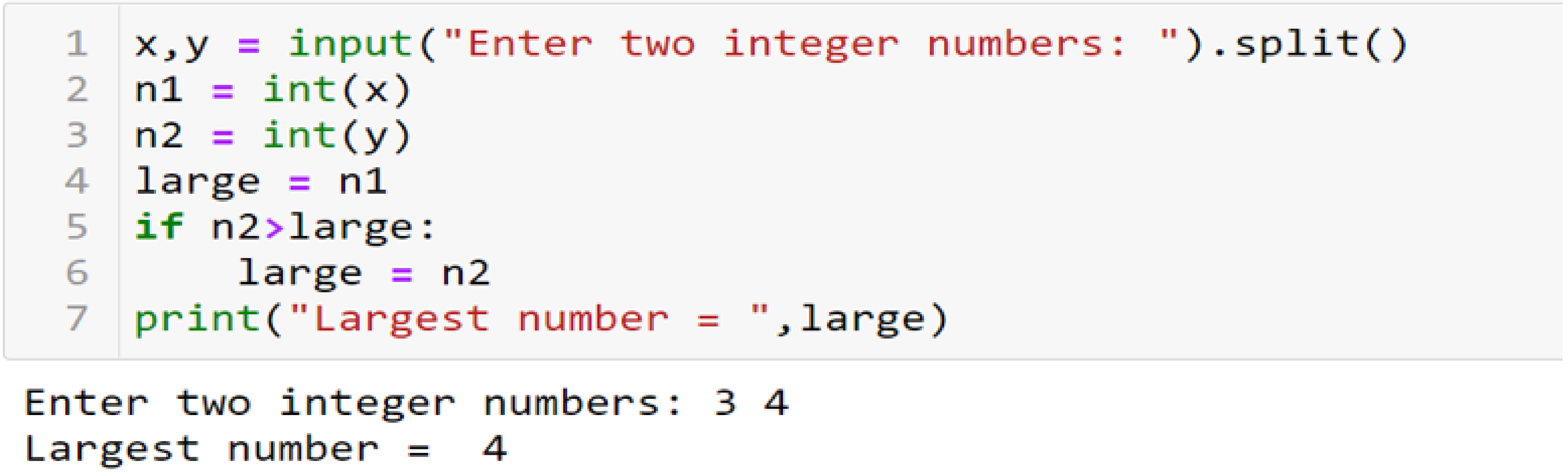
if statement



```
Syntax:

    if cond:
        S1
        S2
        --
        Sn
    Sn+1
```

Fig: Syntax and flow diagram for if statement

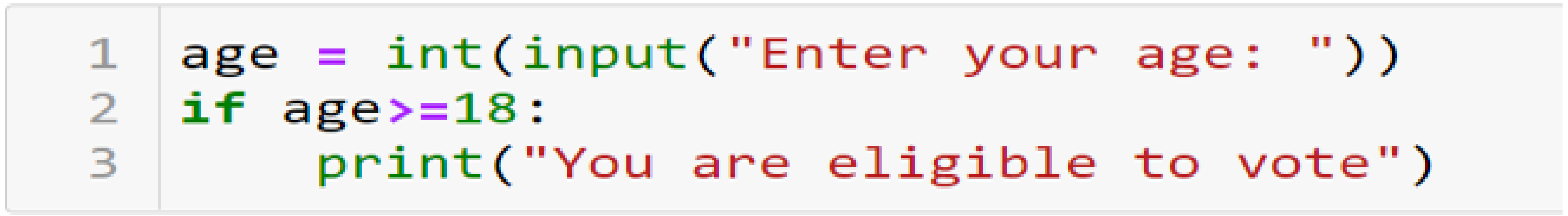
Example1: Python program to find the largest number using if statement.



```python
x,y = input("Enter two integer numbers: ").split()
n1 = int(x)
n2 = int(y)
large = n1
if n2>large:
    large = n2
print("Largest number = ",large)
```

```
Enter two integer numbers: 3 4
Largest number =  4
```

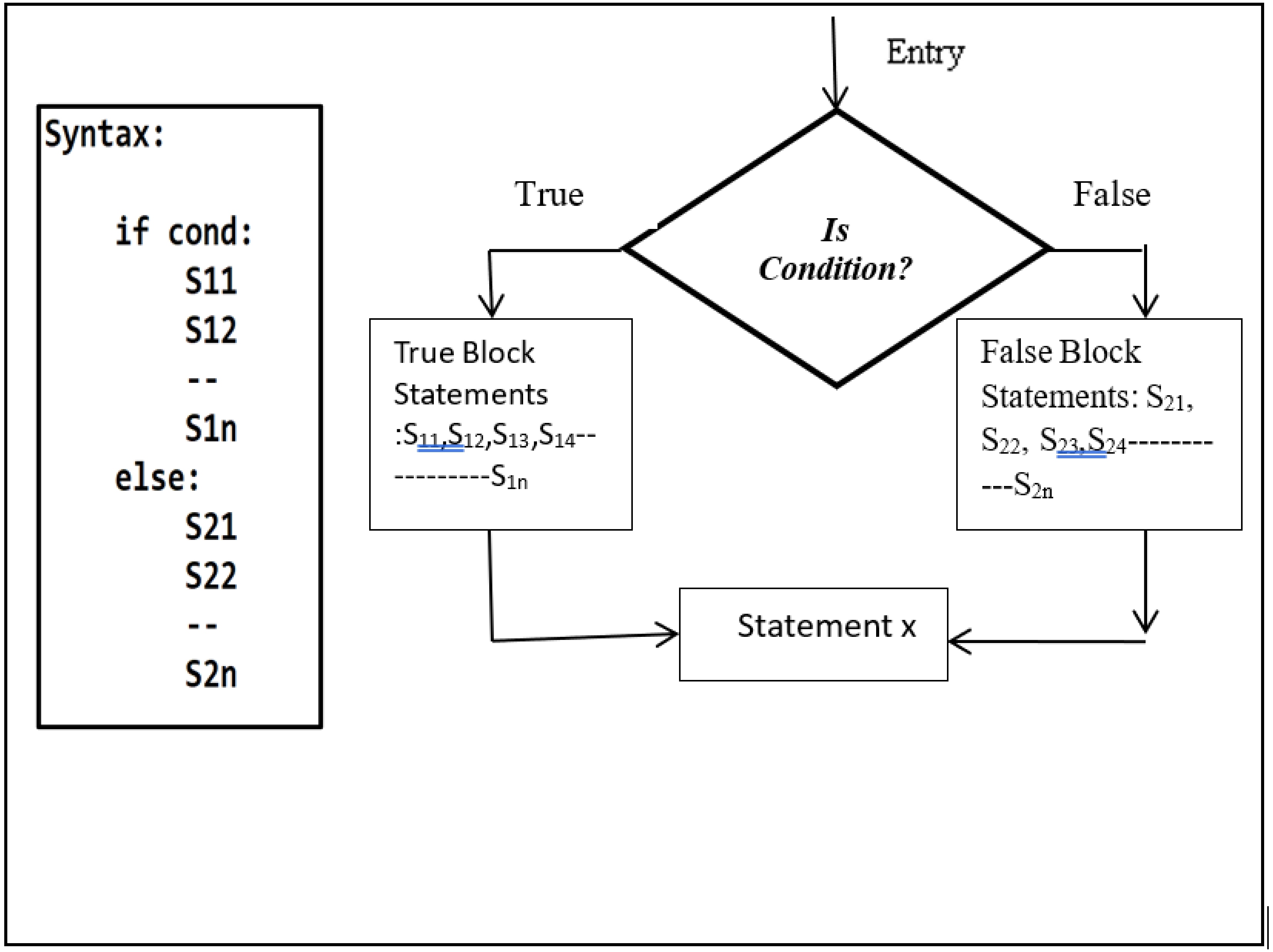
Example2: Python Program to determine whether a person is eligible to vote using if.



```python
age = int(input("Enter your age: "))
if age>=18:
    print("You are eligible to vote")
```

```
Enter your age: 25
You are eligible to vote
```

2. If else statement:

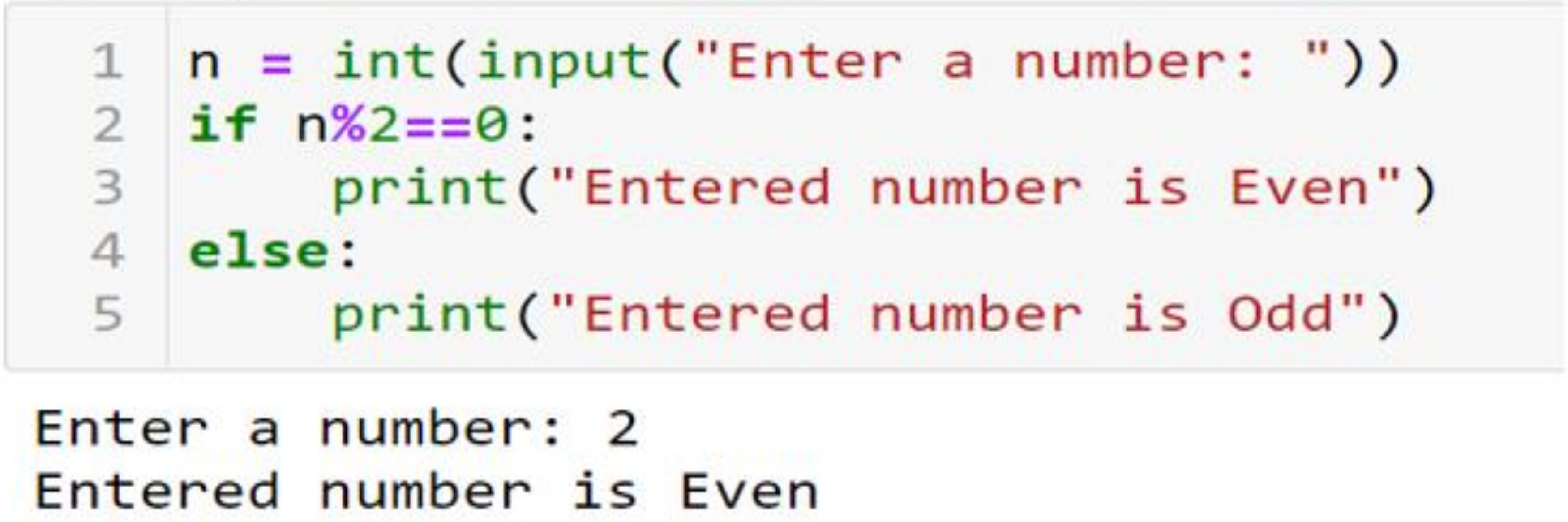


```
Syntax:

    if cond:
        S11
        S12
        --
        S1n
    else:
        S21
        S22
        --
        S2n
```

Entry
Is Condition?
True
False
True Block Statements :S11,S12,S13,S14-----------S1n
False Block Statements: S21, S22, S23,S24-----------S2n
Statement x

Fig: Syntax and flow diagram for if else statement

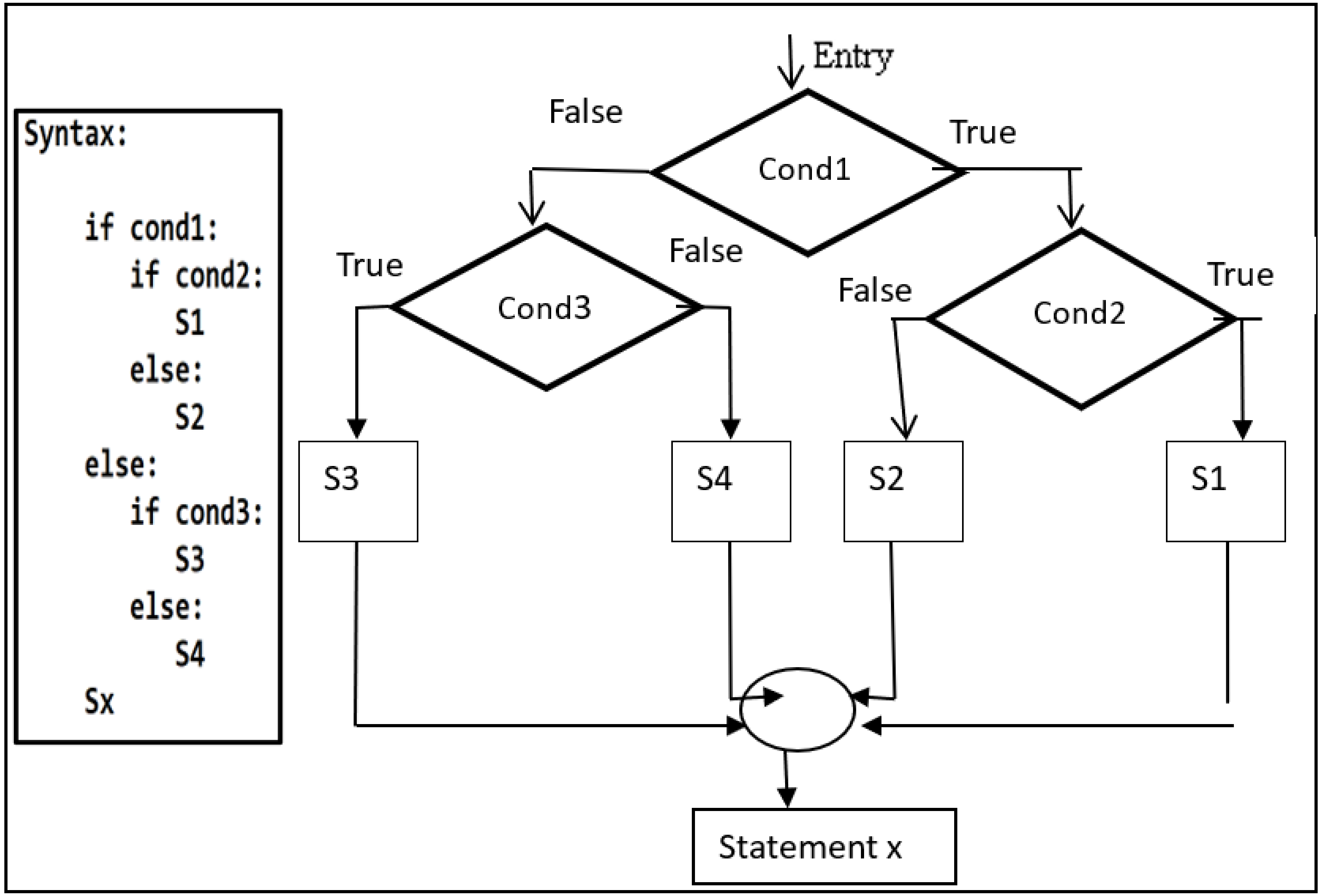
Example: Program to check whether a given number is even or odd using if else.



```python
n = int(input("Enter a number: "))
if n%2==0:
    print("Entered number is Even")
else:
    print("Entered number is Odd")
```

```
Enter a number: 2
Entered number is Even
```

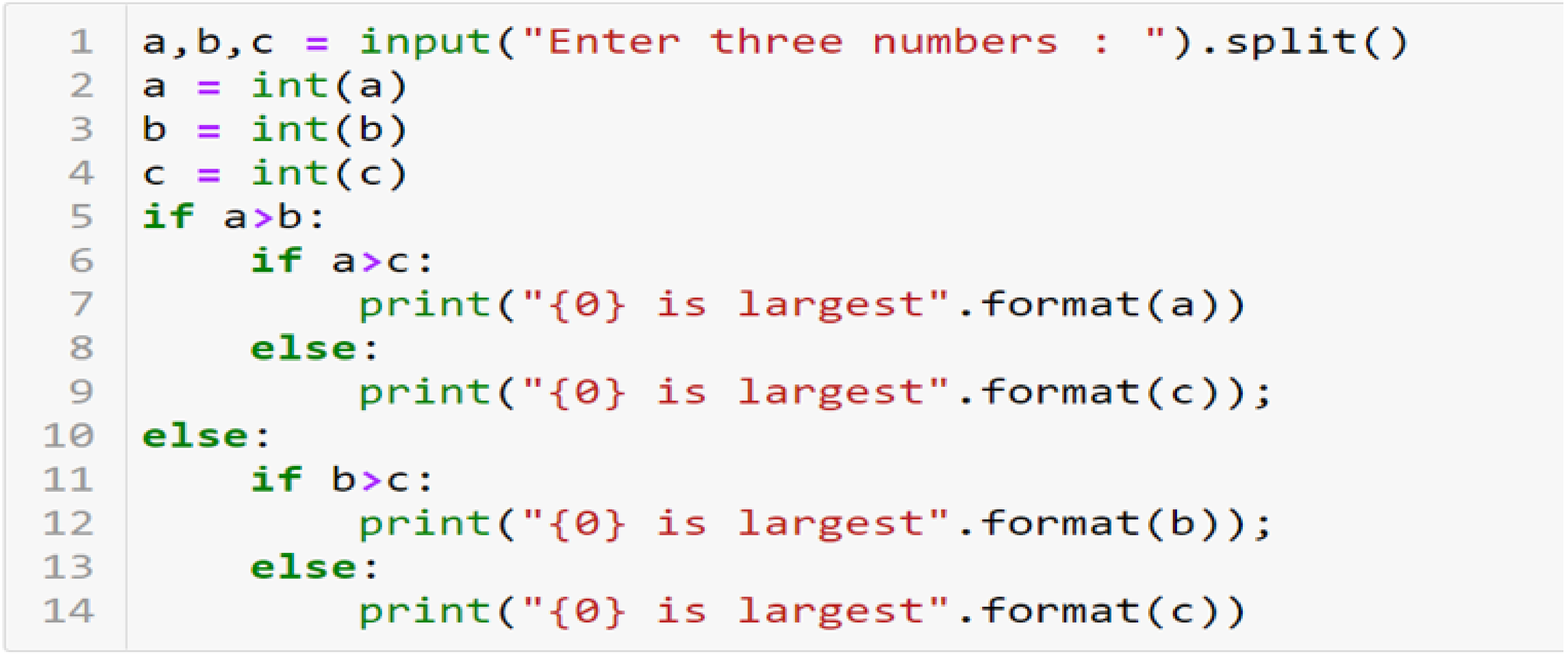
3. Nested if else:


Syntax:

```
    if cond1:
        if cond2:
            S1
        else:
            S2
    else:
        if cond3:
            S3
        else:
            S4
    Sx
```

Fig : Syntax and flow diagram for nested if else statement

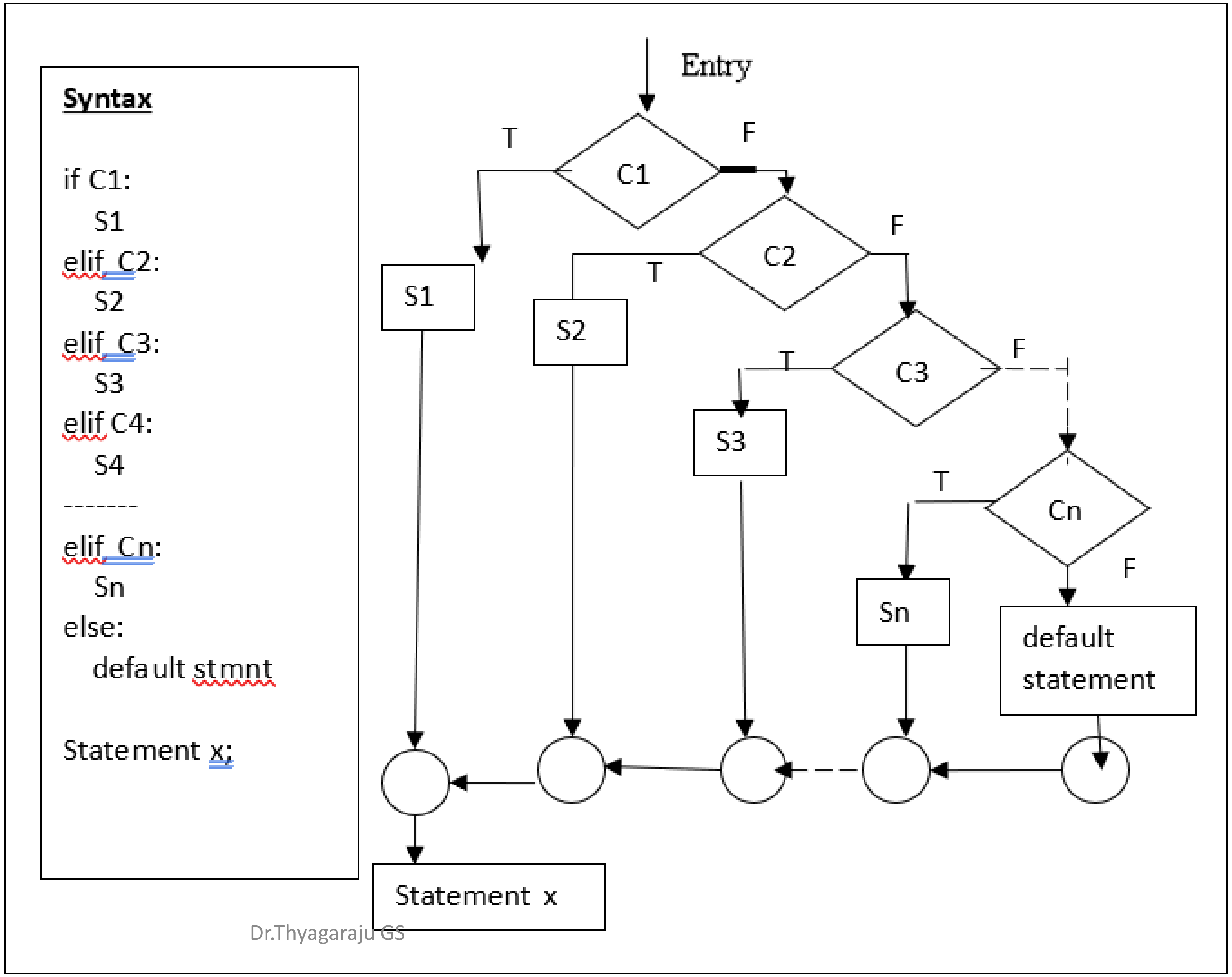
Example:



```
a,b,c = input("Enter three numbers : ").split()
a = int(a)
b = int(b)
c = int(c)
if a>b:
    if a>c:
        print("{0} is largest".format(a))
    else:
        print("{0} is largest".format(c));
else:
    if b>c:
        print("{0} is largest".format(b));
    else:
        print("{0} is largest".format(c))
```

```
Enter three numbers : -2 3 -7
3 is largest
```

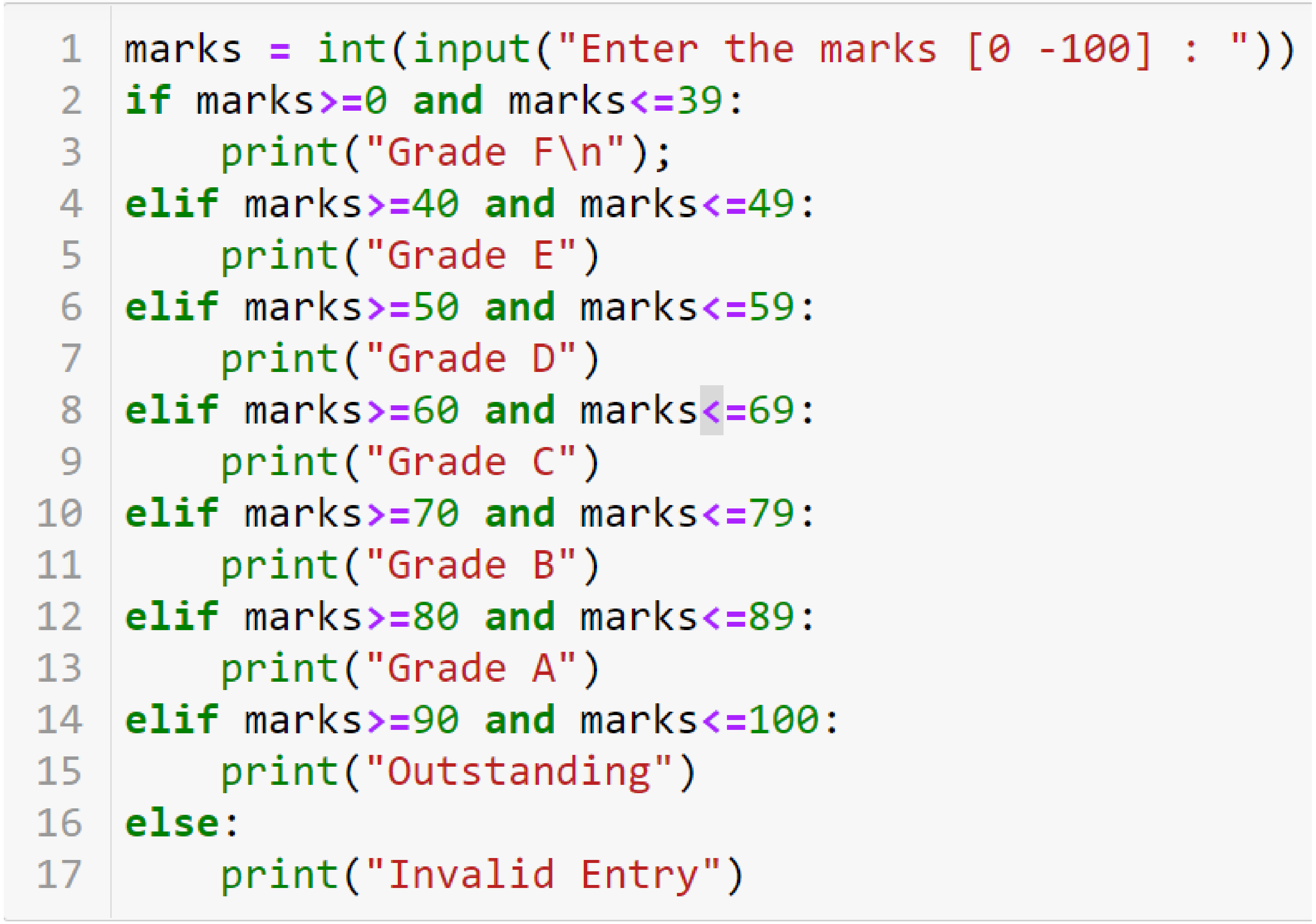
4. *if – elif* ladder:



Syntax

```
if C1:
   S1
elif C2:
   S2
elif C3:
   S3
elif C4:
   S4
-------
elif Cn:
   Sn
else:
   default stmnt

Statement x;
```

Example



```python
marks = int(input("Enter the marks [0 -100] : "))
if marks>=0 and marks<=39:
    print("Grade F\n");
elif marks>=40 and marks<=49:
    print("Grade E")
elif marks>=50 and marks<=59:
    print("Grade D")
elif marks>=60 and marks<=69:
    print("Grade C")
elif marks>=70 and marks<=79:
    print("Grade B")
elif marks>=80 and marks<=89:
    print("Grade A")
elif marks>=90 and marks<=100:
    print("Outstanding")
else:
    print("Invalid Entry")
```

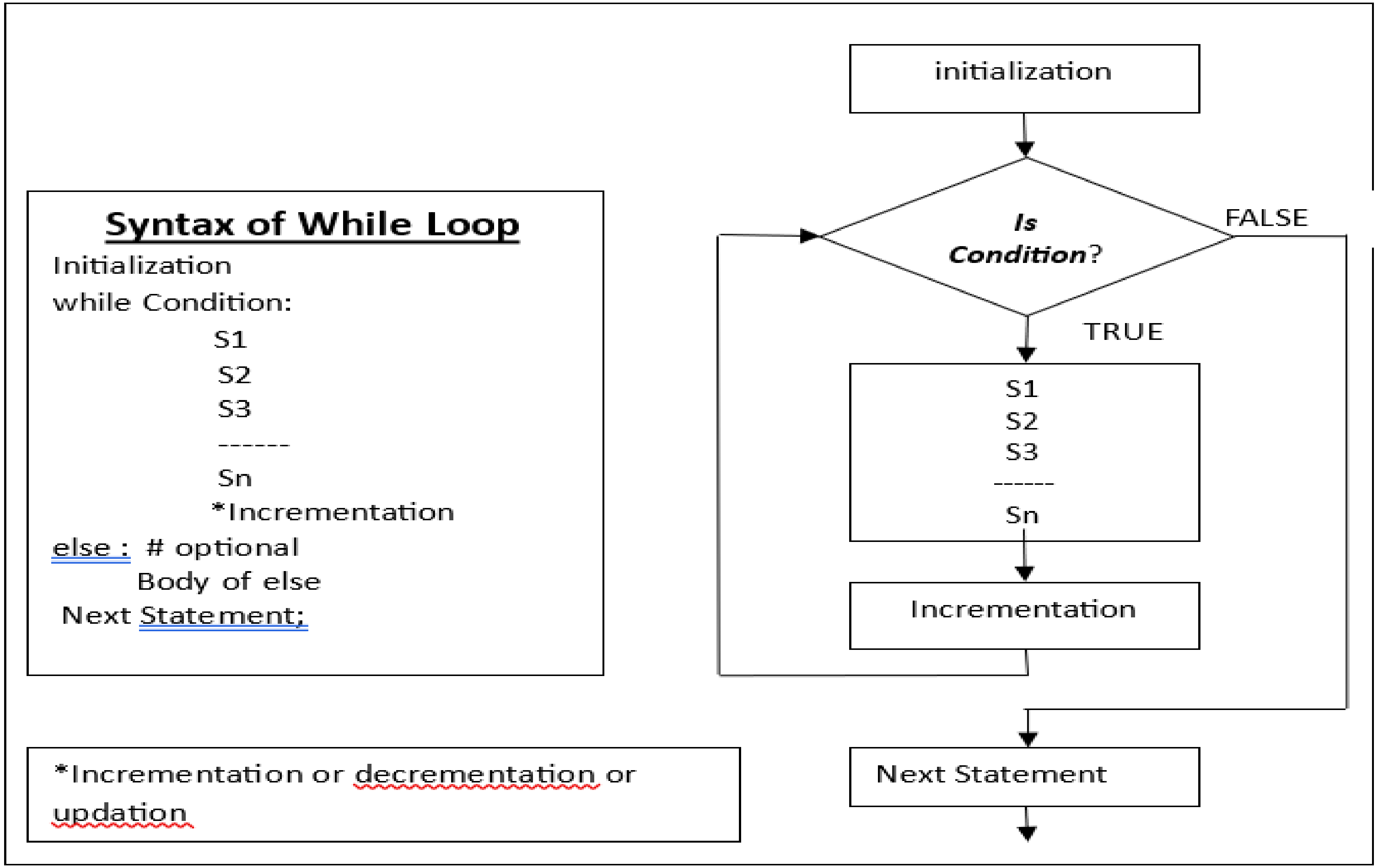
```
Enter the marks [0 -100] : 90
Outstanding
```

Types of Loops Supported in Python

- The Python Language supports the following two looping operations:

1. **The while statement**
2. **The for statement**

while Statement



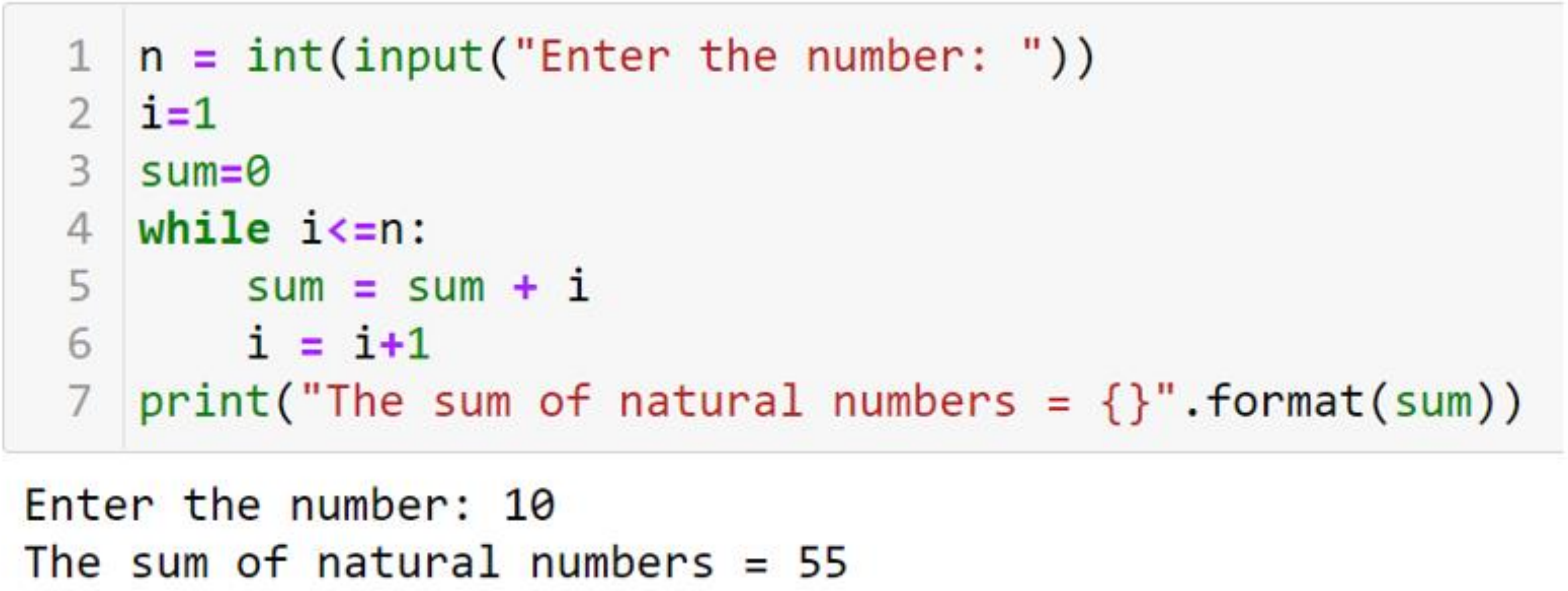
Syntax of While Loop

```
Initialization
while Condition:
          S1
          S2
          S3
          ------
          Sn
          *Incrementation
else :  # optional
     Body of else
 Next Statement;
```

*Incrementation or decrementation or updation

Fig: Flow diagram for Exit Controlled Loop

Example: Program to find the sum of n natural numbers using while loop.



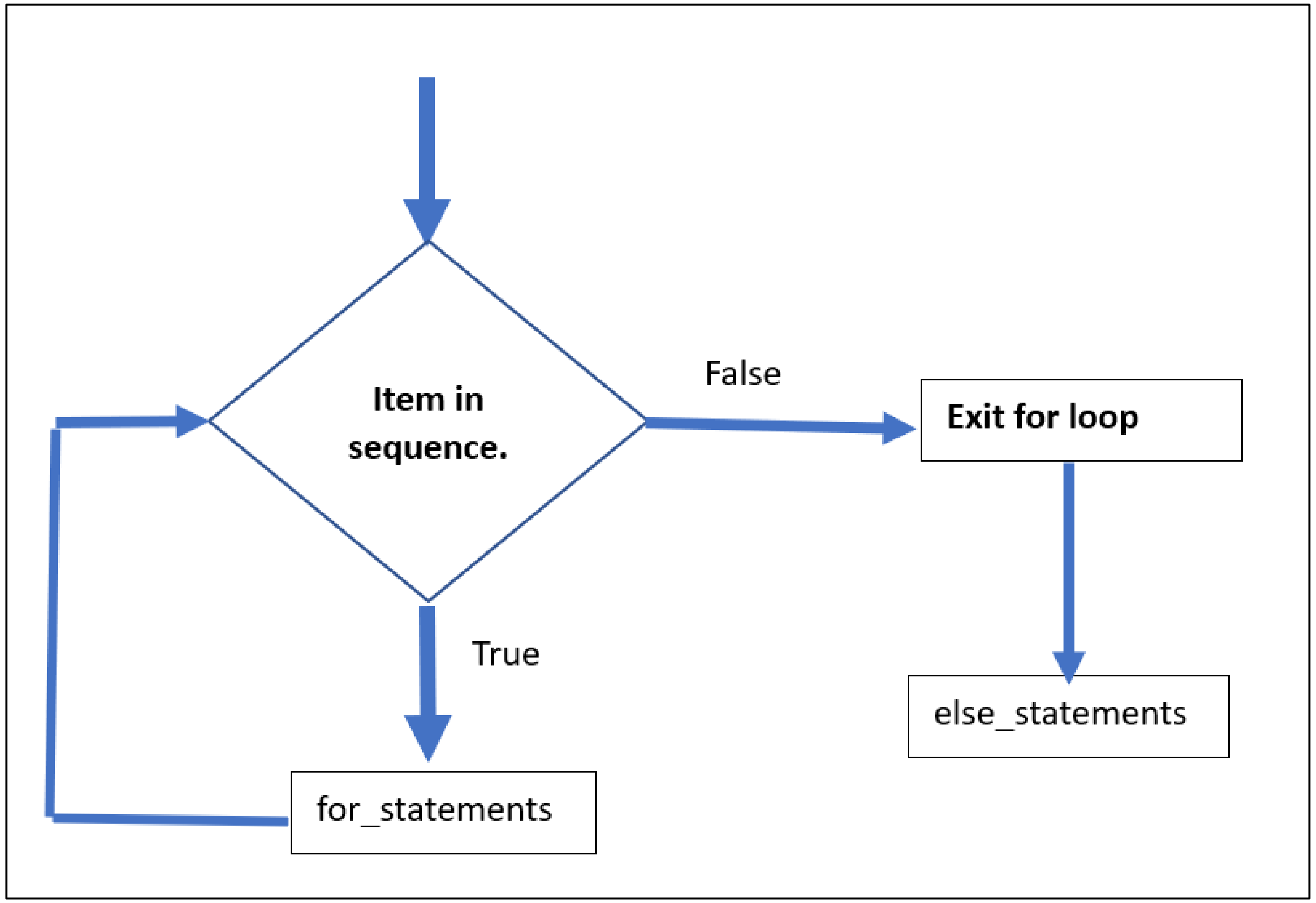
```python
n = int(input("Enter the number: "))
i=1
sum=0
while i<=n:
    sum = sum + i
    i = i+1
print("The sum of natural numbers = {}".format(sum))
```

```
Enter the number: 10
The sum of natural numbers = 55
```

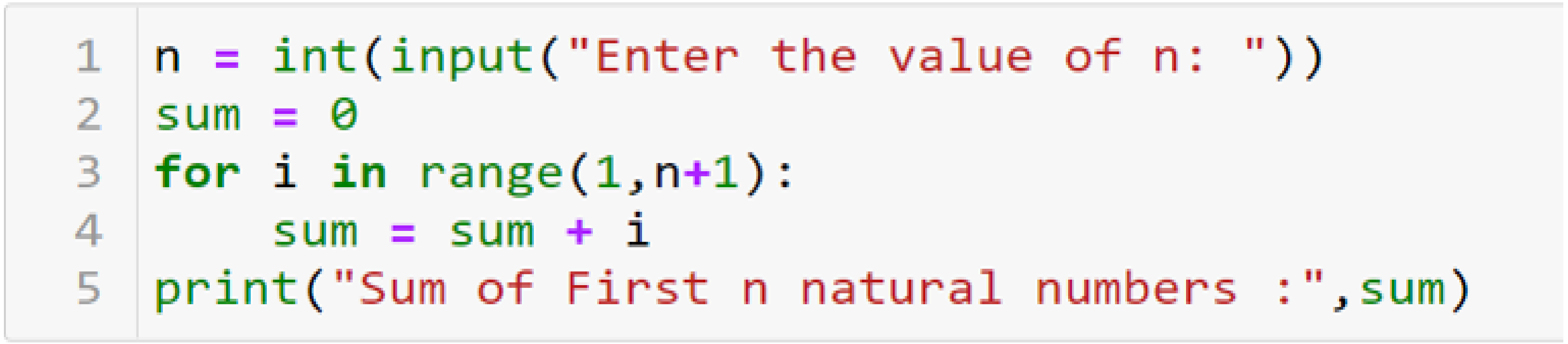
for Loop

Syntax:

```
    for item in iterable:
        for_statements
    else: # optional
        else_statements
```



Example: Python Program to find the sum of natural numbers upto n.



```
n = int(input("Enter the value of n: "))
sum = 0
for i in range(1,n+1):
    sum = sum + i
print("Sum of First n natural numbers :",sum)
```

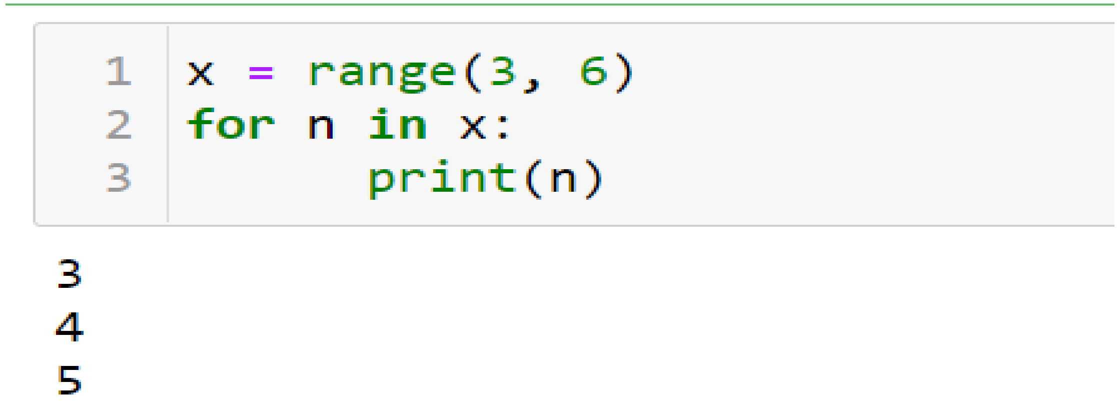
```
Enter the value of n: 10
Sum of First n natural numbers : 55
```

range ( ) function:

The range() function returns a sequence of numbers, starting from 0 to a specified number ,incrementing each time by 1.

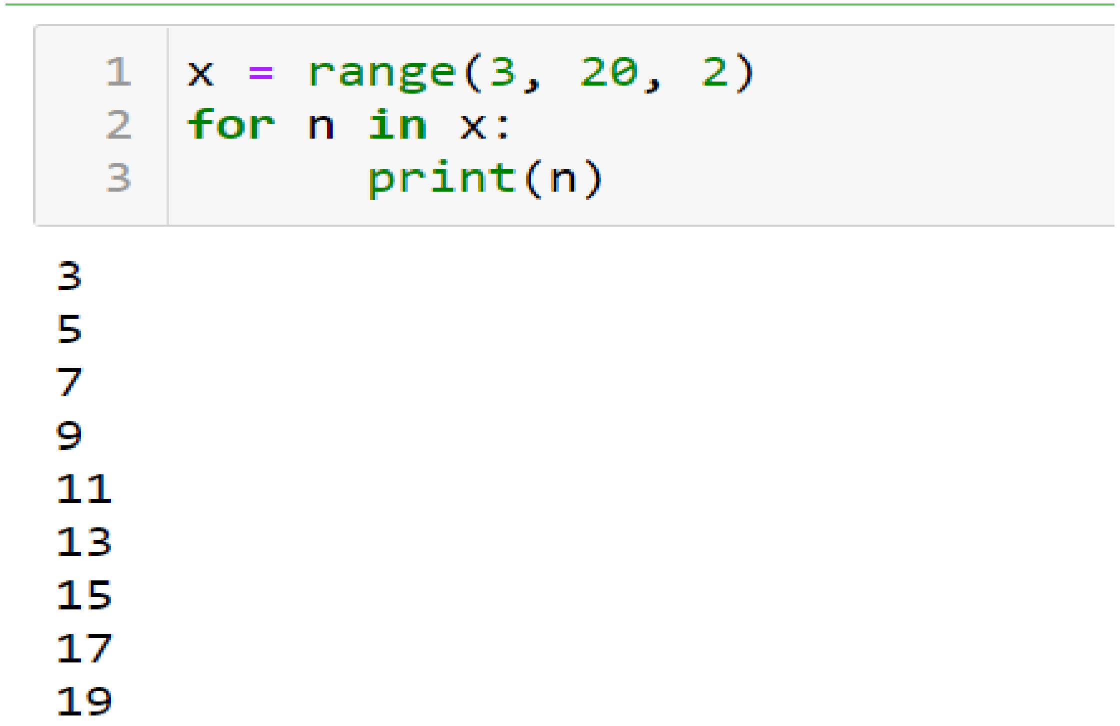
Syntax :

range(start,step,stop)



```
x = range(3, 6)
for n in x:
        print(n)
```

```
3
4
5
```



```
x = range(3, 20, 2)
for n in x:
        print(n)
```

```
3
5
7
9
11
13
15
17
19
```

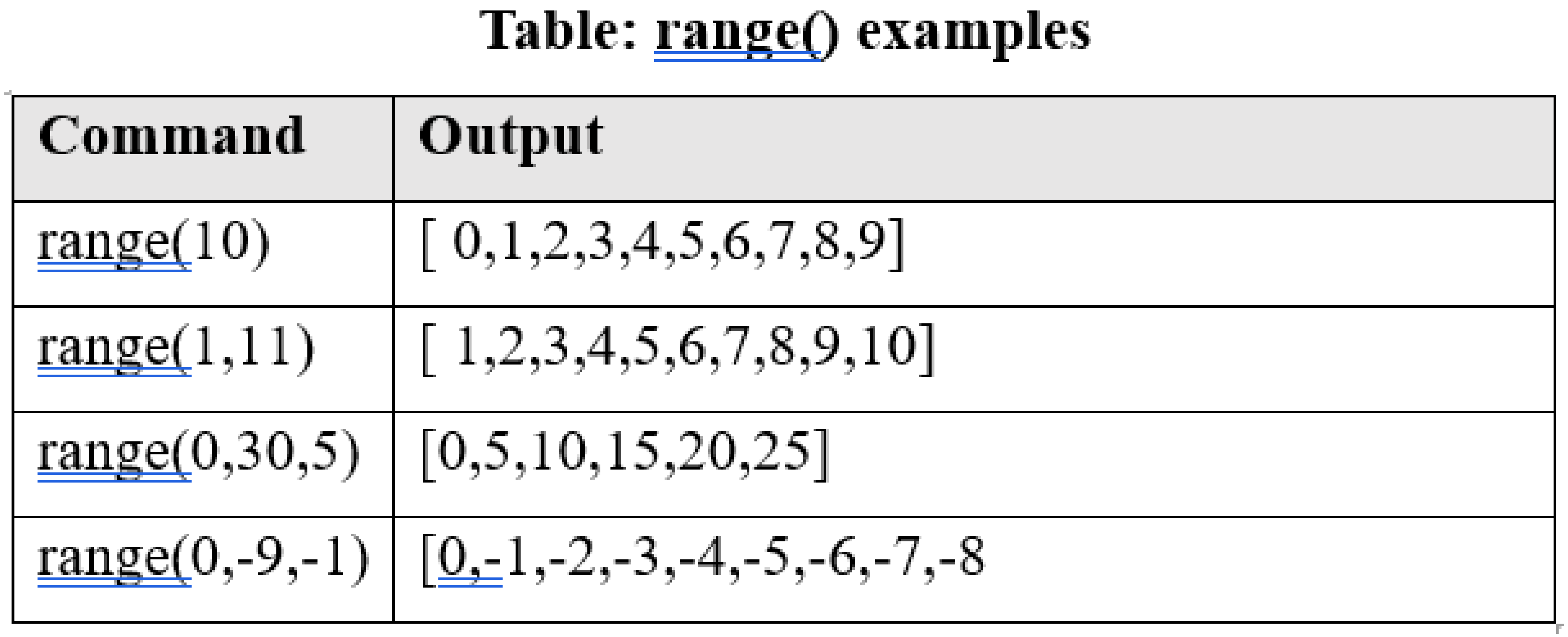
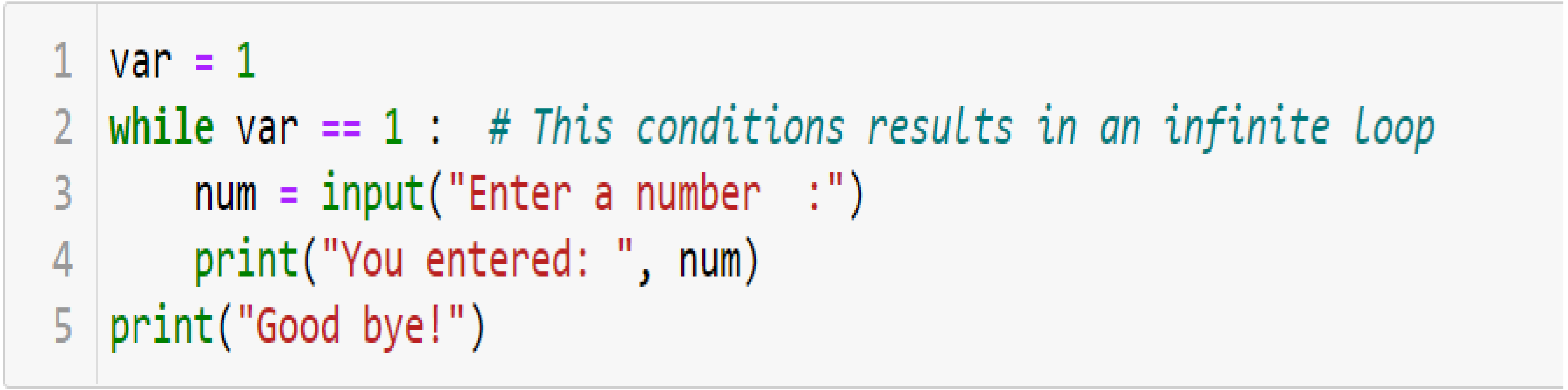


Table: range() examples

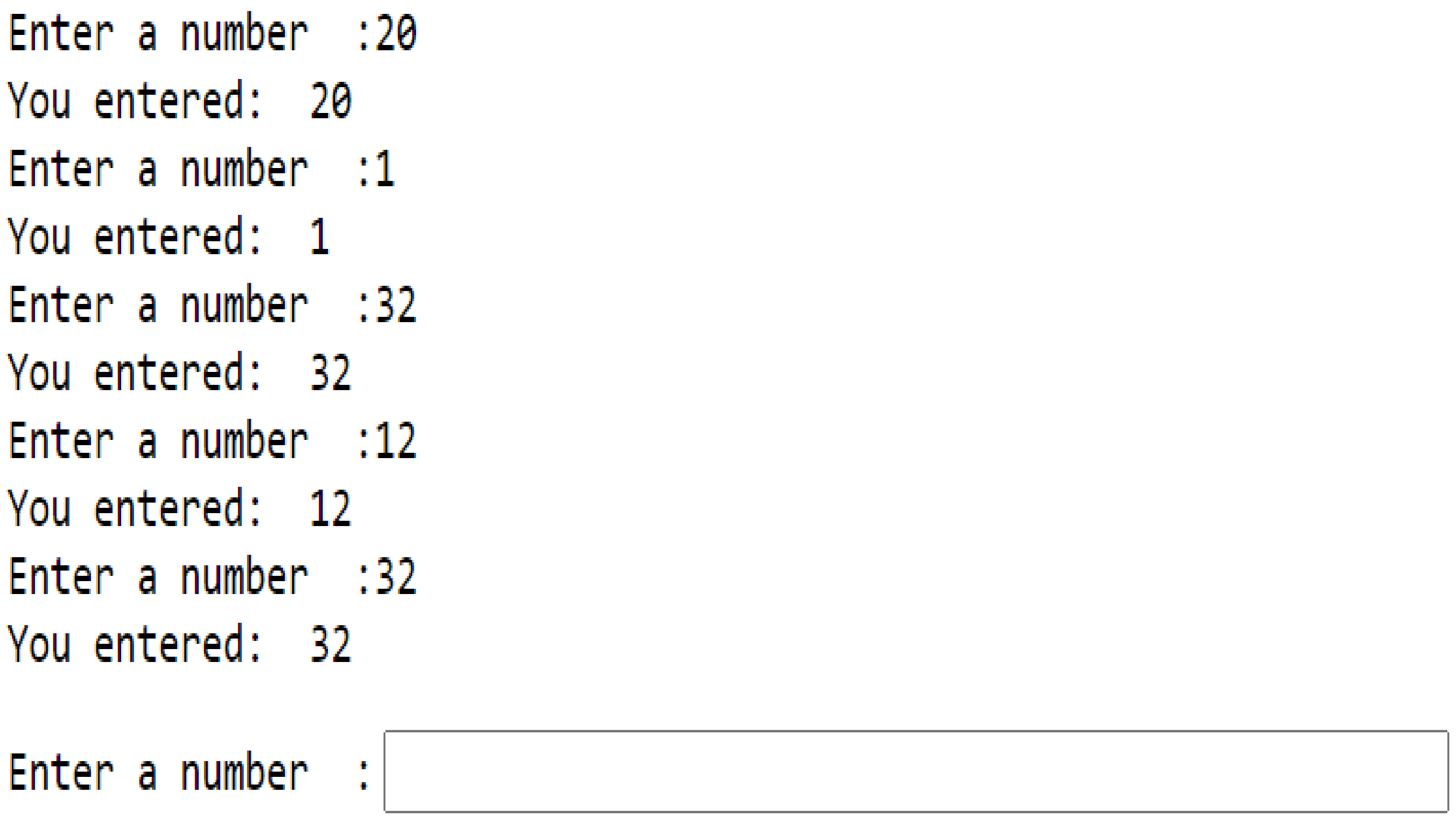
Command	Output
range(10)	[0,1,2,3,4,5,6,7,8,9]
range(1,11)	[1,2,3,4,5,6,7,8,9,10]
range(0,30,5)	[0,5,10,15,20,25]
range(0,-9,-1)	[0,-1,-2,-3,-4,-5,-6,-7,-8

Infinite Loop

A loop becomes infinite loop if a condition never becomes FALSE. You must use caution when using while loops because of the possibility that this condition never resolves to a FALSE value. This results in a loop that never ends. Such a loop is called an infinite loop.

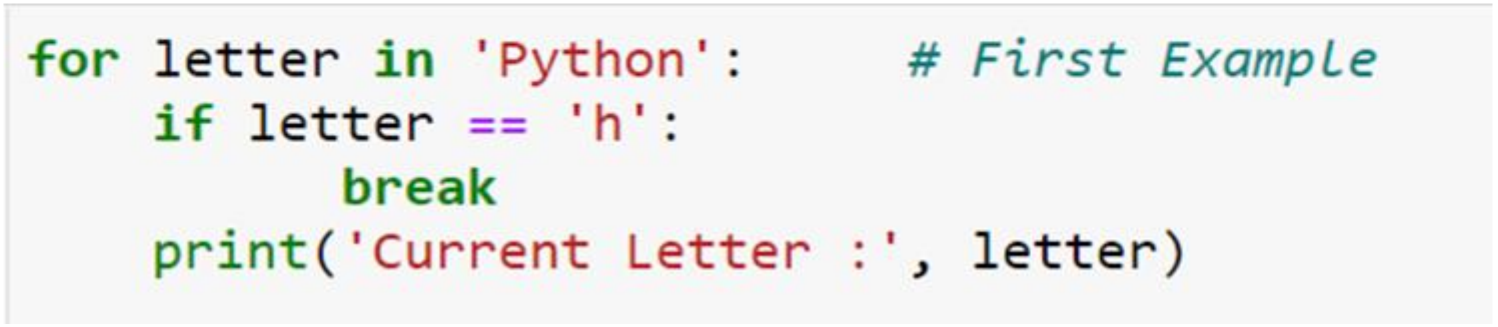


```python
var = 1
while var == 1 :  # This conditions results in an infinite loop
    num = input("Enter a number  :")
    print("You entered: ", num)
print("Good bye!")
```

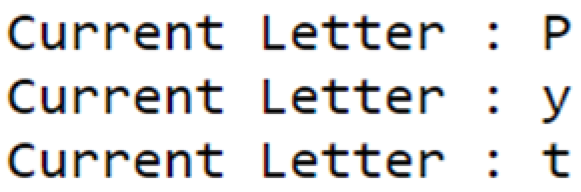


```
Enter a number  :20
You entered:  20
Enter a number  :1
You entered:  1
Enter a number  :32
You entered:  32
Enter a number  :12
You entered:  12
Enter a number  :32
You entered:  32

Enter a number  :
```

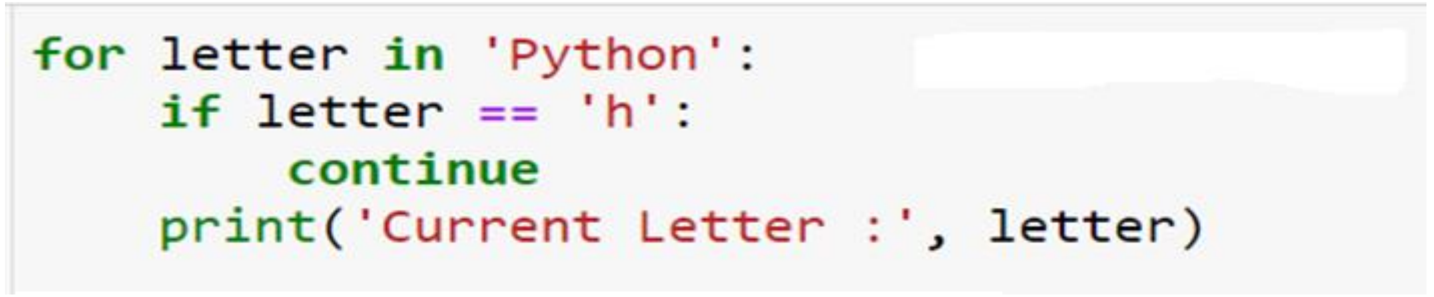
break and continue



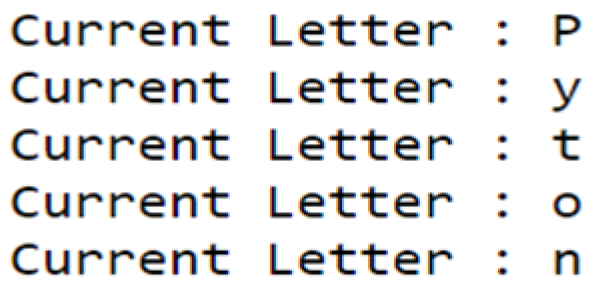
```
for letter in 'Python':     # First Example
    if letter == 'h':
        break
    print('Current Letter :', letter)
```



```
Current Letter : P
Current Letter : y
Current Letter : t
```



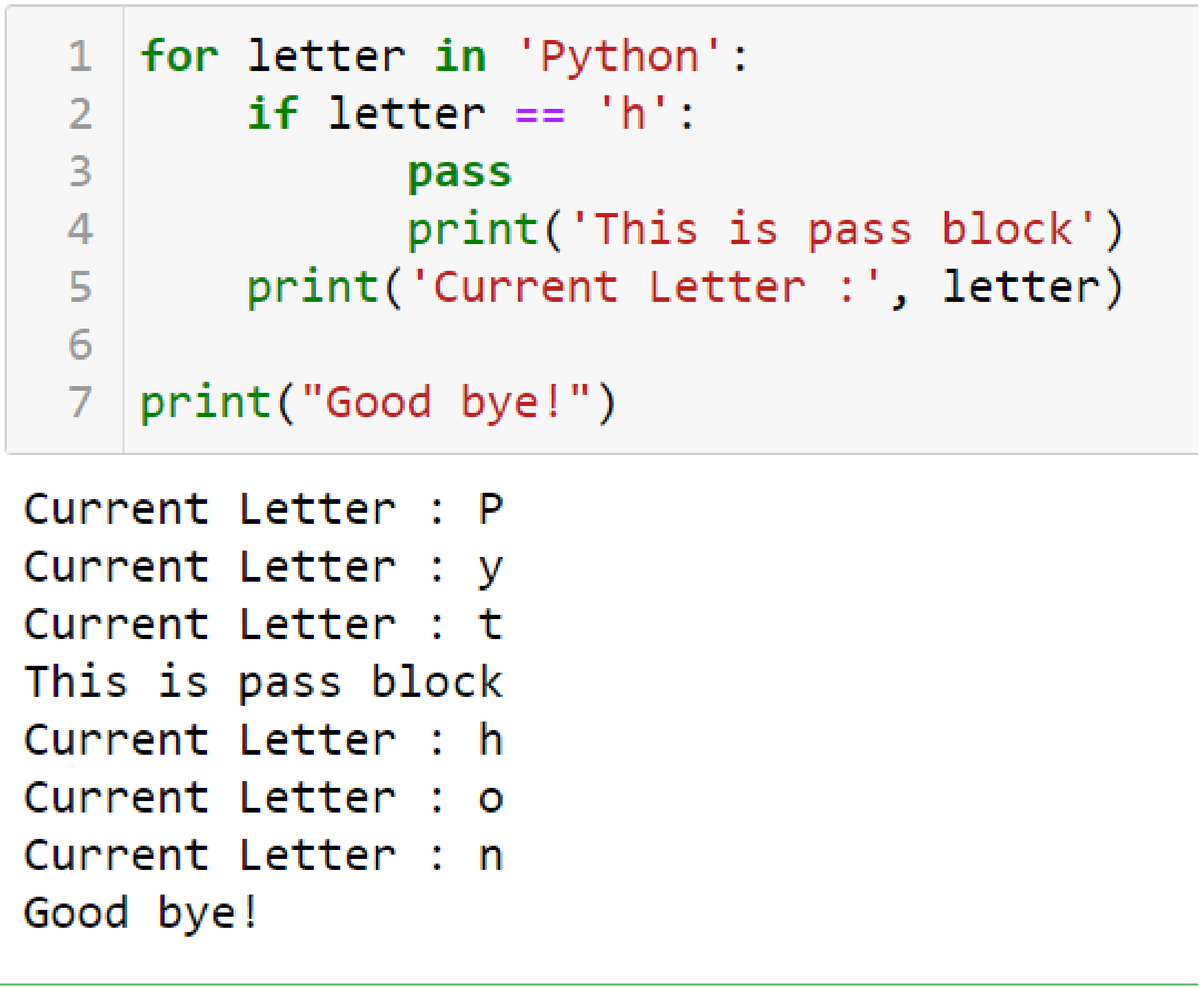
```
for letter in 'Python':
    if letter == 'h':
        continue
    print('Current Letter :', letter)
```



```
Current Letter : P
Current Letter : y
Current Letter : t
Current Letter : o
Current Letter : n
```

The Pass Statement :

- The **pass** statement in Python is used when a statement is required syntactically but you do not want any command or code to execute.
- The **pass** statement is a *null* operation; nothing happens when it executes. The **pass** is also useful in places where your code will eventually go, but has not been written yet (e.g., in stubs for example):



```python
for letter in 'Python':
    if letter == 'h':
        pass
        print('This is pass block')
    print('Current Letter :', letter)

print("Good bye!")
```

```
Current Letter : P
Current Letter : y
Current Letter : t
This is pass block
Current Letter : h
Current Letter : o
Current Letter : n
Good bye!
```

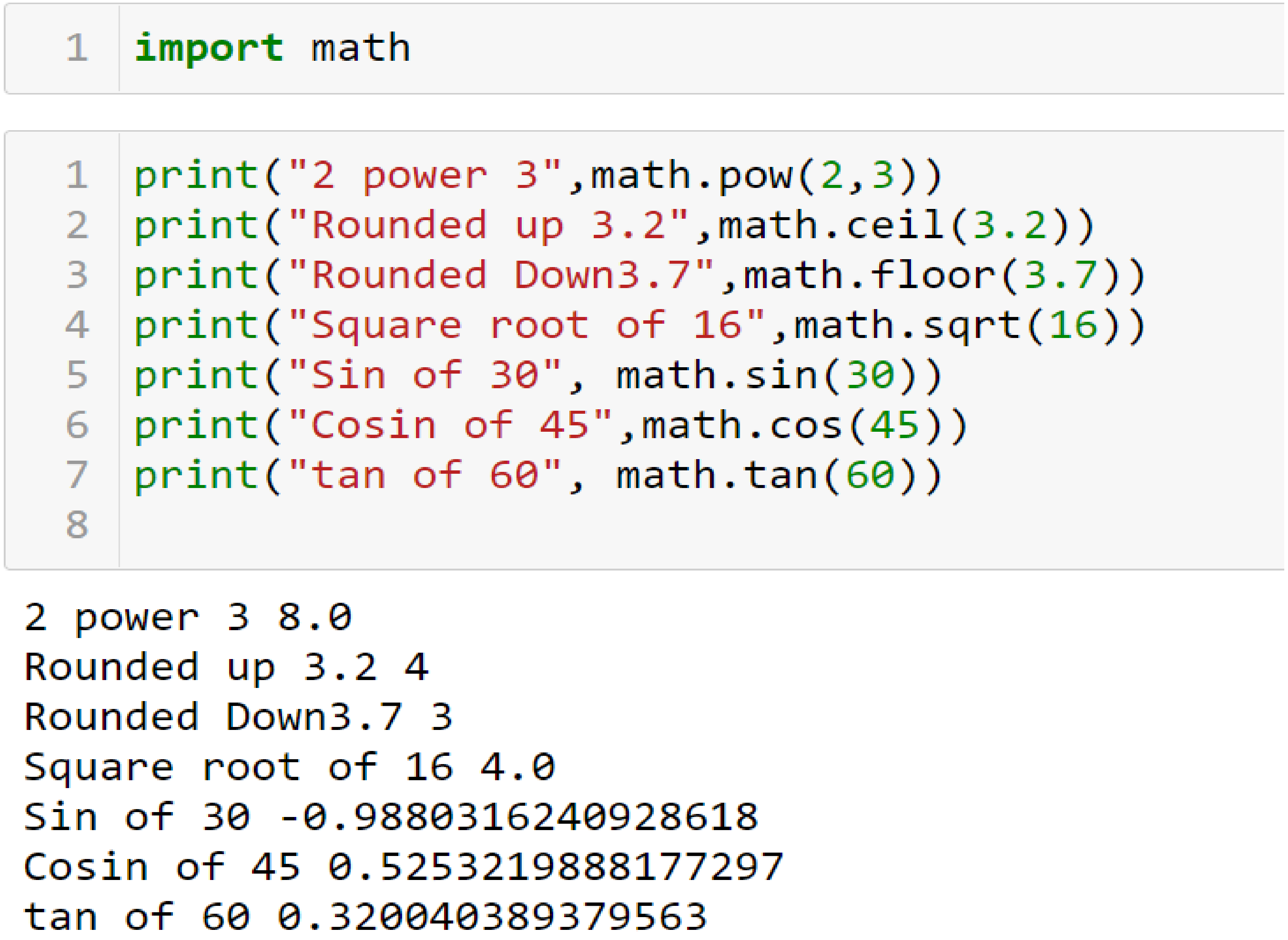
Importing Modules

- Each module is a Python program that contains a related group of functions that can be embedded in your programs. For example, the **math** module has mathematics related functions, the **random** module has random number–related functions, and so on.

In code, an import statement consists of the following:

- **The import keyword**
- **The name of the module**
- **Optionally, more module names, as long as they are separated by commas**

Importing math

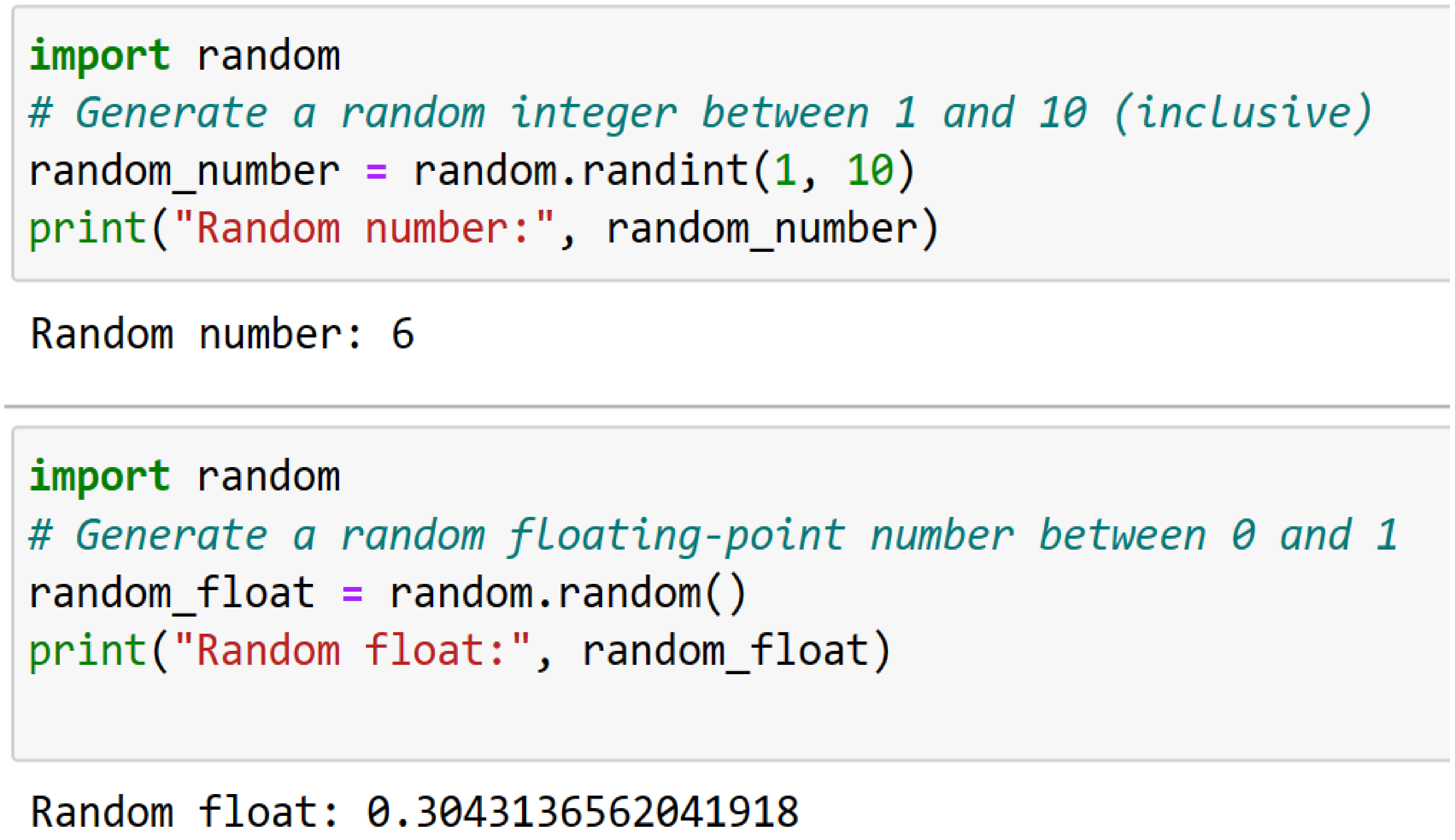


```python
import math
```

```python
print("2 power 3",math.pow(2,3))
print("Rounded up 3.2",math.ceil(3.2))
print("Rounded Down3.7",math.floor(3.7))
print("Square root of 16",math.sqrt(16))
print("Sin of 30", math.sin(30))
print("Cosin of 45",math.cos(45))
print("tan of 60", math.tan(60))

```

```
2 power 3 8.0
Rounded up 3.2 4
Rounded Down3.7 3
Square root of 16 4.0
Sin of 30 -0.9880316240928618
Cosin of 45 0.5253219888177297
tan of 60 0.320040389379563
```

Importing random



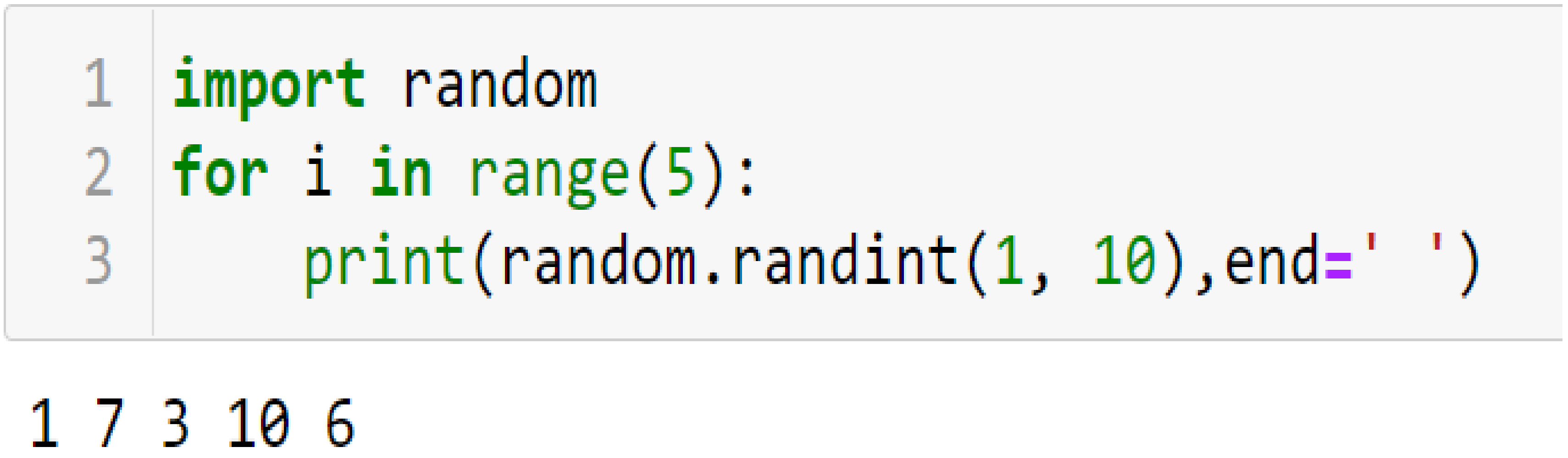
```python
import random
# Generate a random integer between 1 and 10 (inclusive)
random_number = random.randint(1, 10)
print("Random number:", random_number)
```

```
Random number: 6
```

```python
import random
# Generate a random floating-point number between 0 and 1
random_float = random.random()
print("Random float:", random_float)
```

```
Random float: 0.3043136562041918
```

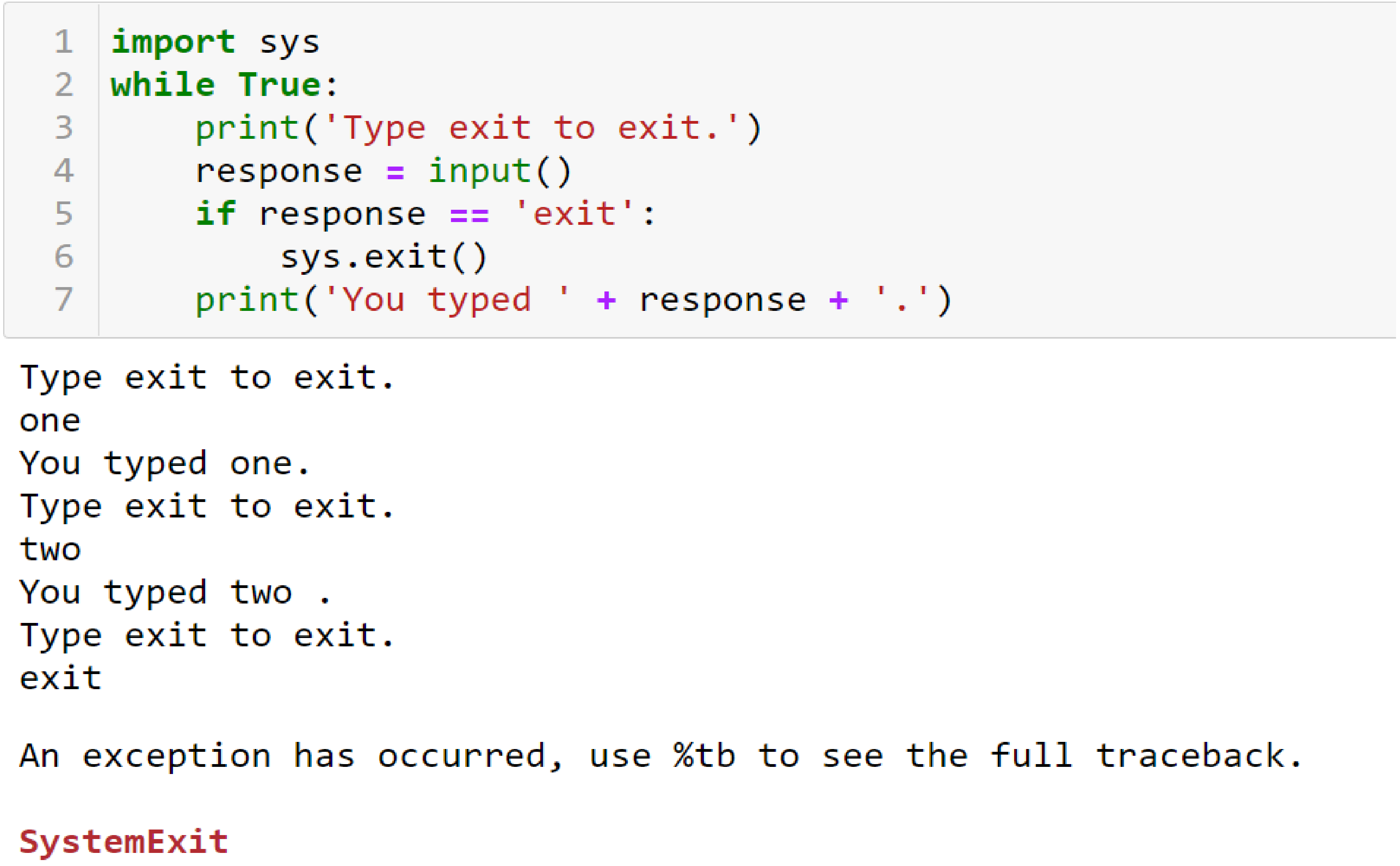
Importing random



```python
import random
for i in range(5):
    print(random.randint(1, 10),end=' ')
```

```
1 7 3 10 6
```

Ending a Program Early with sys.exit ()



```python
import sys
while True:
    print('Type exit to exit.')
    response = input()
    if response == 'exit':
        sys.exit()
    print('You typed ' + response + '.')
```

```
Type exit to exit.
one
You typed one.
Type exit to exit.
two
You typed two .
Type exit to exit.
exit

An exception has occurred, use %tb to see the full traceback.

SystemExit
```

1.3 Functions

- Syllabus : def Statements with Parameters, Return Values and return Statements, The None Value, Keyword Arguments and print(), Local and Global Scope, The global Statement, Exception Handling, A Short Program: Guess the Number

What is Function?

- A function is a group (or block) of statements that perform a specific task .
- Functions run only when it is called.
- One can pass data into the function in the form of parameters.
- Function can also return data as a result.

Types of Functions

1. User Defined
2. Built in

User defined Functions

- In Python, user-defined functions are functions that are created by the programmer to perform specific tasks. These functions are defined using the **def** keyword followed by the function name, parentheses for optional parameters, and a colon to start the function block.

Syntax for Defining a function:

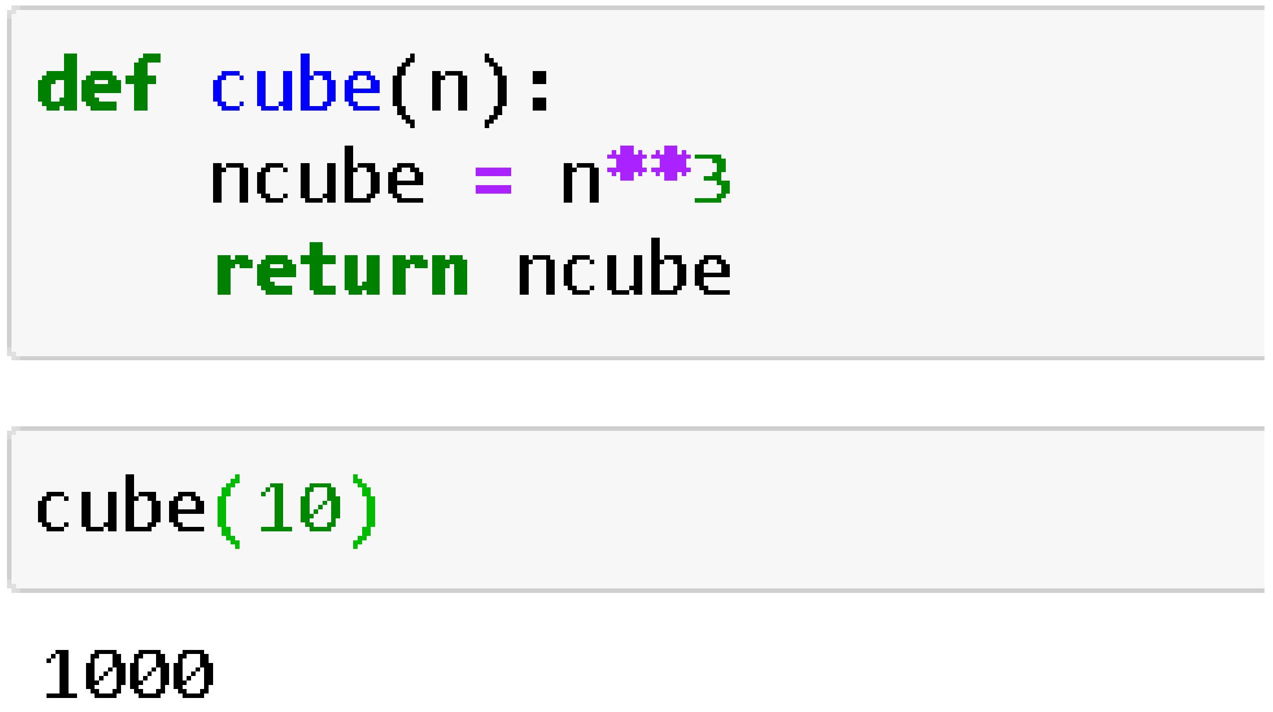
Function is defined using def keyword in Python.

```
def   fun_name(comma_seprated_parameter_ list):
          stmt_1
          -------
          stmt_n
          return stmt
```

Syntax for calling a function:

```
fun_name(parameter list)
```

Example



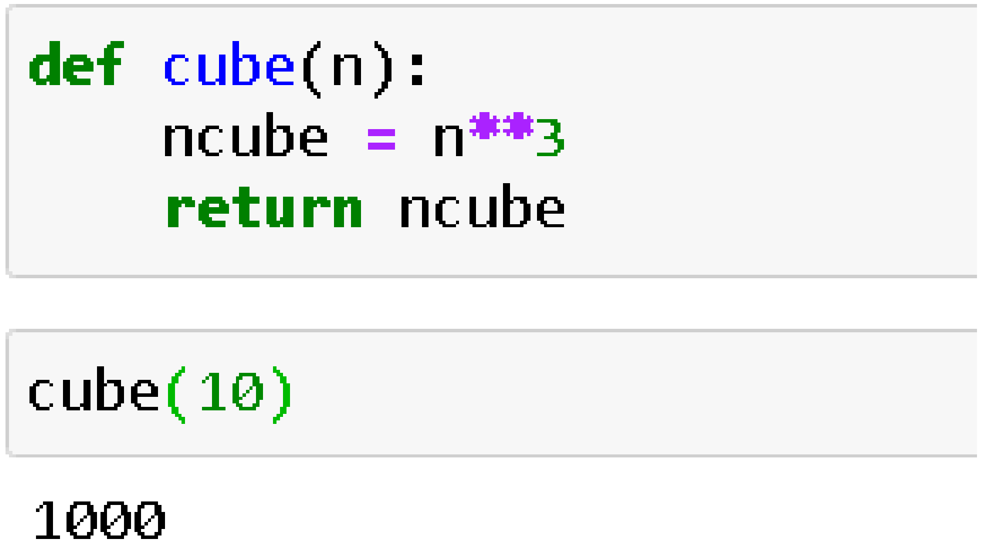
```
def cube(n):
    ncube = n**3
    return ncube
```

```
cube(10)
```

```
1000
```

Parameters and arguments

- **Parameters** are temporary variable names within functions.
- The **argument** can be thought of as the value that is assigned to that temporary variable.
- '**n**' here is the ***parameter*** for the function '**cube**'. This means that anywhere we see '**n**' within the function will act as a placeholder until number is passed an argument.
- Here **10** is the ***argument***.
- Parameters are used in ***function definition*** and arguments are used in ***function call***.

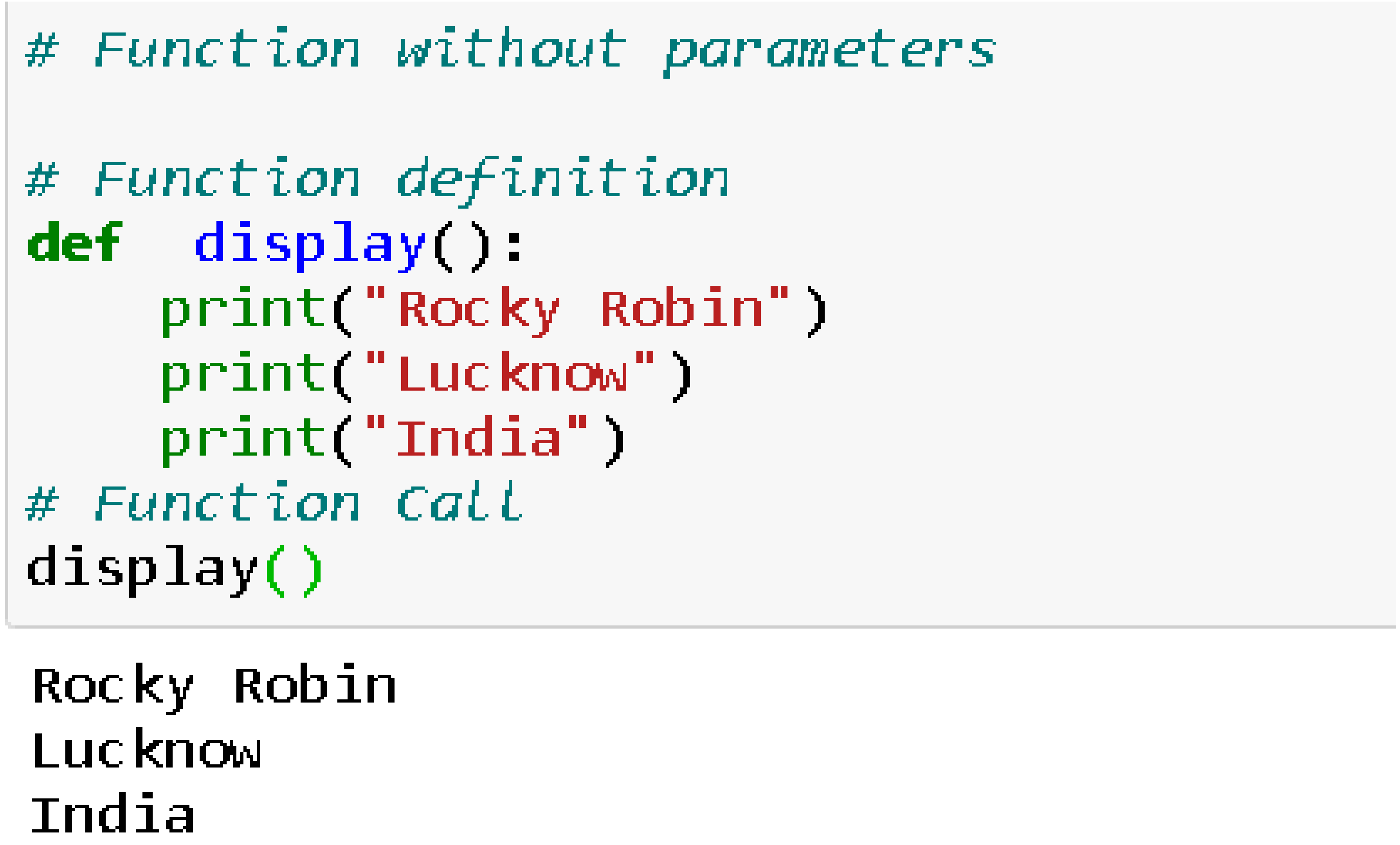


```python
def cube(n):
    ncube = n**3
    return ncube
```

```python
cube(10)
```

```
1000
```

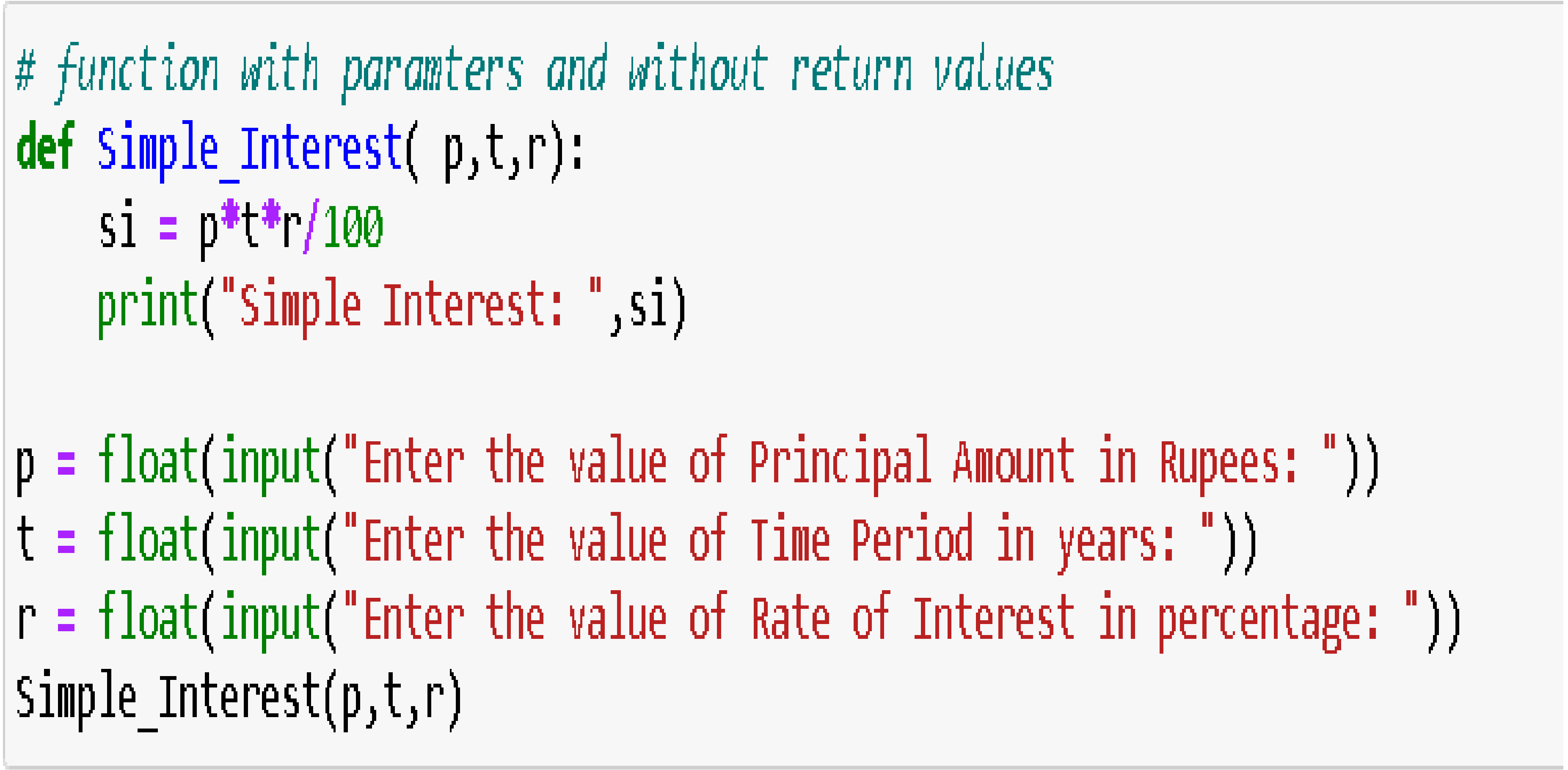
Function without parameters



```
# Function without parameters

# Function definition
def  display():
     print("Rocky Robin")
     print("Lucknow")
     print("India")
# Function Call
display()
```

```
Rocky Robin
Lucknow
India
```

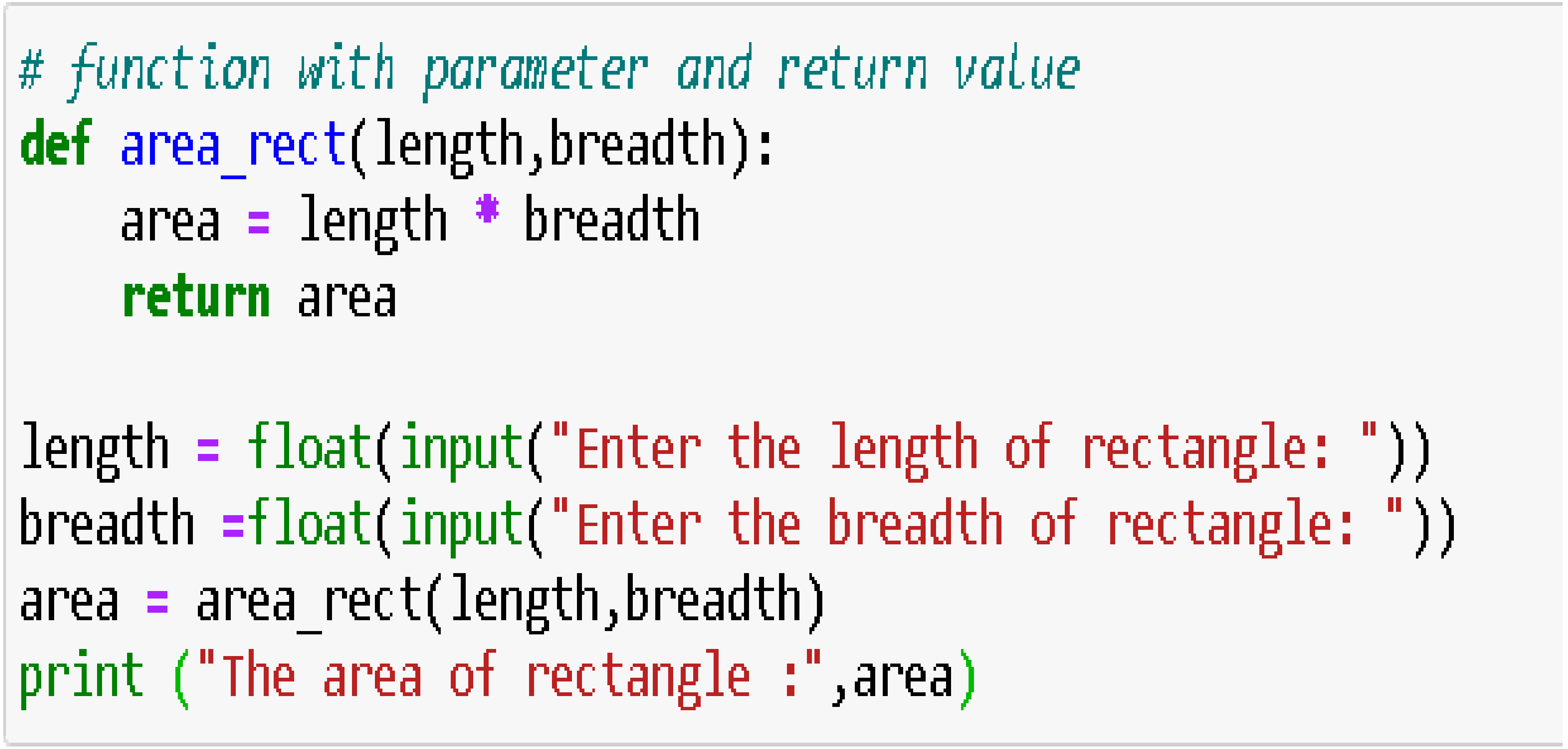
Function with parameters but without returning values



```
# function with paramters and without return values
def Simple_Interest( p,t,r):
    si = p*t*r/100
    print("Simple Interest: ",si)

p = float(input("Enter the value of Principal Amount in Rupees: "))
t = float(input("Enter the value of Time Period in years: "))
r = float(input("Enter the value of Rate of Interest in percentage: "))
Simple_Interest(p,t,r)
```

```
Enter the value of Principal Amount in Rupees: 1000
Enter the value of Time Period in years: 2.5
Enter the value of Rate of Interest in percentage: 1.5
Simple Interest:  37.5
```

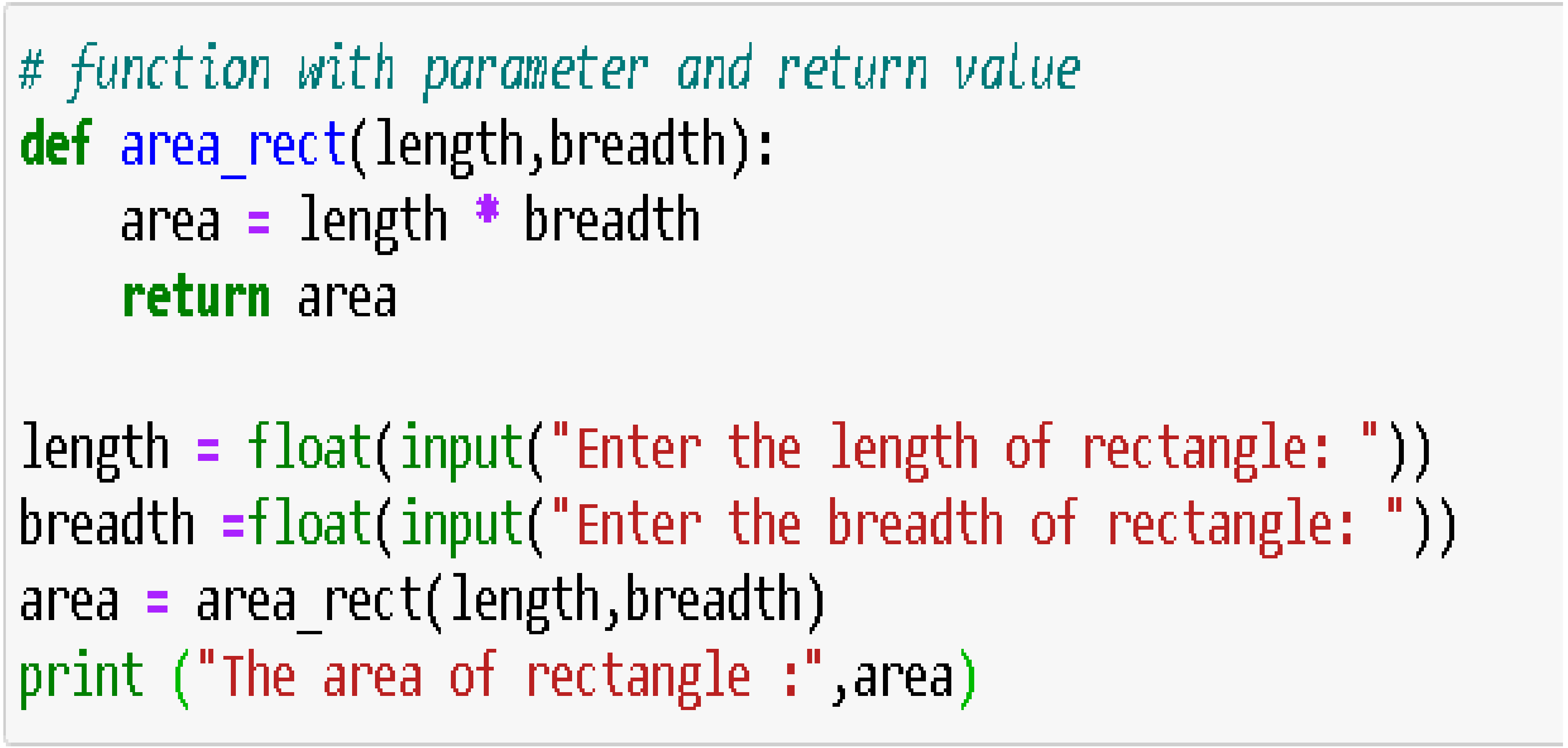
Function with parameters and return values



```
# function with parameter and return value
def area_rect(length,breadth):
    area = length * breadth
    return area

length = float(input("Enter the length of rectangle: "))
breadth =float(input("Enter the breadth of rectangle: "))
area = area_rect(length,breadth)
print ("The area of rectangle :",area)
```

```
Enter the length of rectangle: 10
Enter the breadth of rectangle: 20
The area of rectangle : 200.0
```

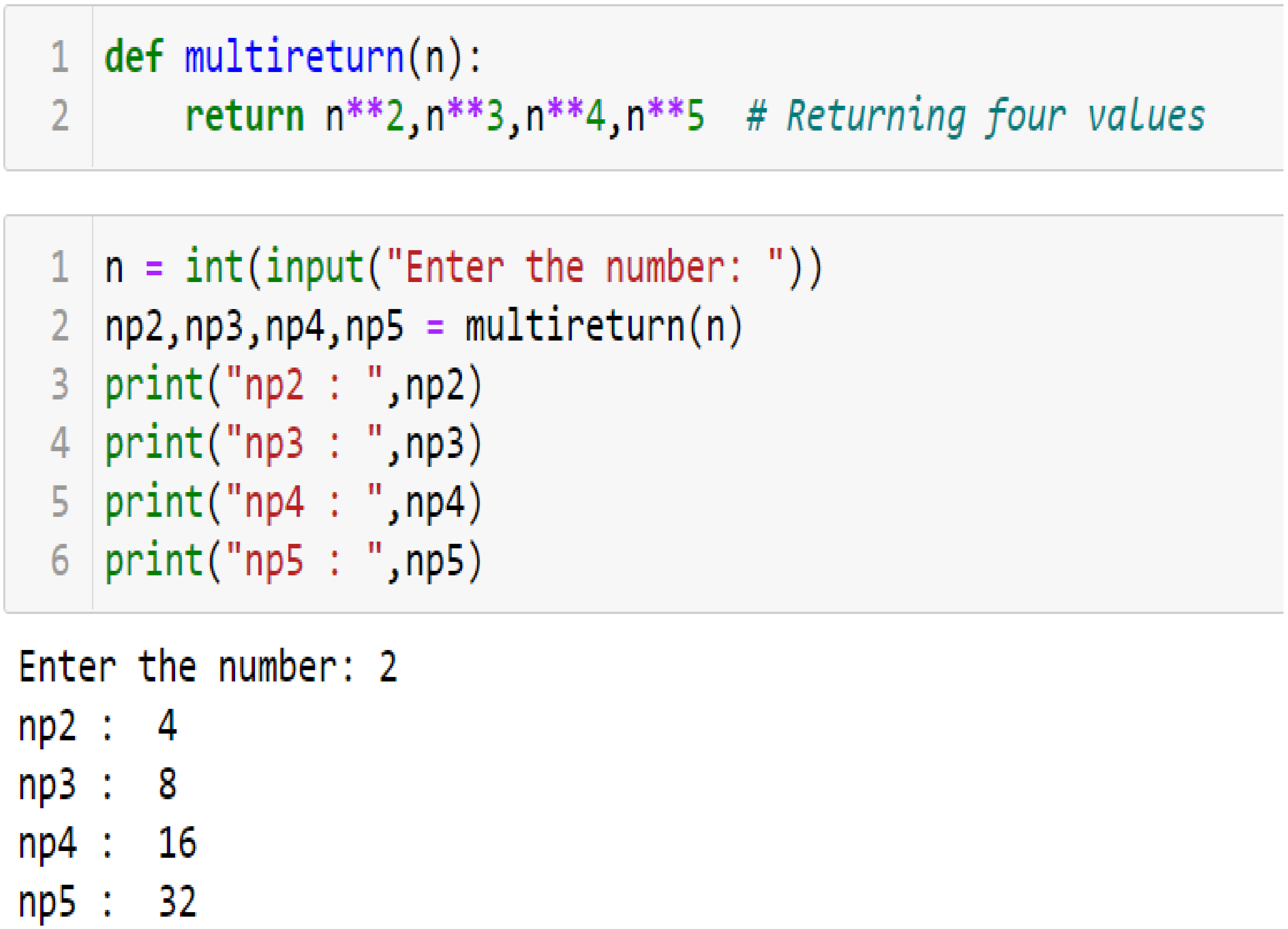
Function with parameters and return values



```python
# function with parameter and return value
def area_rect(length,breadth):
    area = length * breadth
    return area

length = float(input("Enter the length of rectangle: "))
breadth =float(input("Enter the breadth of rectangle: "))
area = area_rect(length,breadth)
print ("The area of rectangle :",area)
```

```
Enter the length of rectangle: 10
Enter the breadth of rectangle: 20
The area of rectangle : 200.0
```

Function which return multiple values



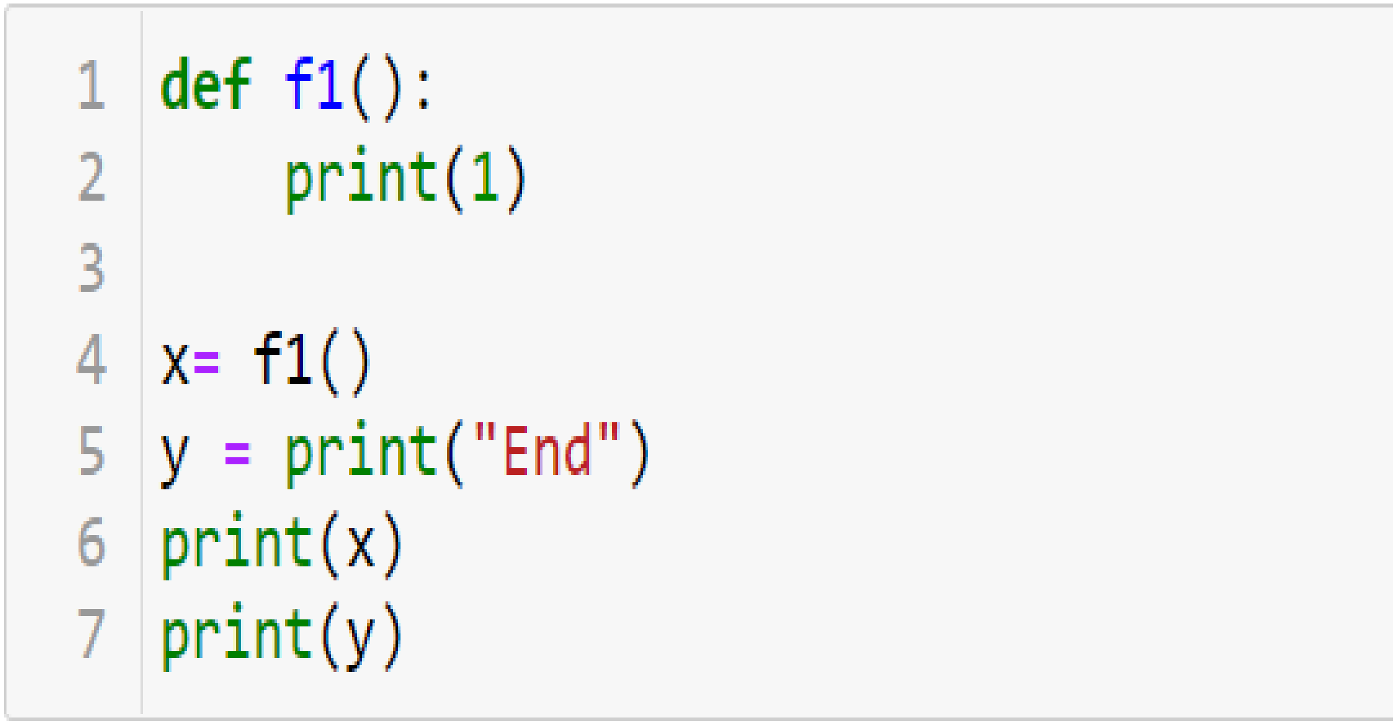
```python
def multireturn(n):
    return n**2,n**3,n**4,n**5  # Returning four values
```

```python
n = int(input("Enter the number: "))
np2,np3,np4,np5 = multireturn(n)
print("np2 : ",np2)
print("np3 : ",np3)
print("np4 : ",np4)
print("np5 : ",np5)
```

```
Enter the number: 2
np2 :  4
np3 :  8
np4 :  16
np5 :  32
```

The None-Value

- In Python there is a value called **None**, which represents the absence of a value. **None** is the only value of the None Type data type.
- (Other programming languages might call this value null, nil, or undefined.)
- Just like the Boolean True and False values, None must be typed with a capital N.



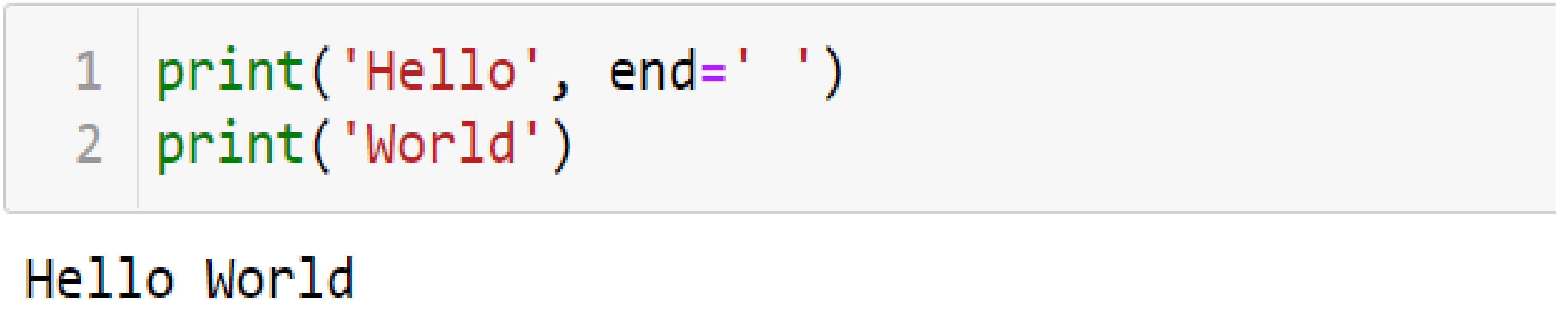
```python
def f1():
    print(1)

x= f1()
y = print("End")
print(x)
print(y)
```

```
1
End
None
None
```

Keyword Arguments and print()

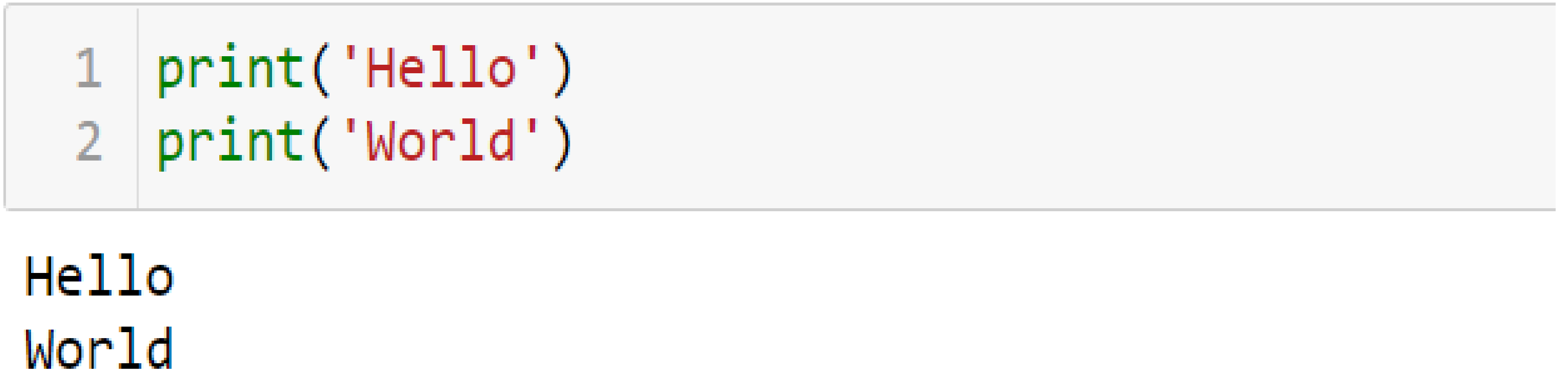
Keyword arguments are identified by the keyword put before them in the function call. Keyword arguments are often used for optional parameters.

Examples : ***end*** and ***sep***.



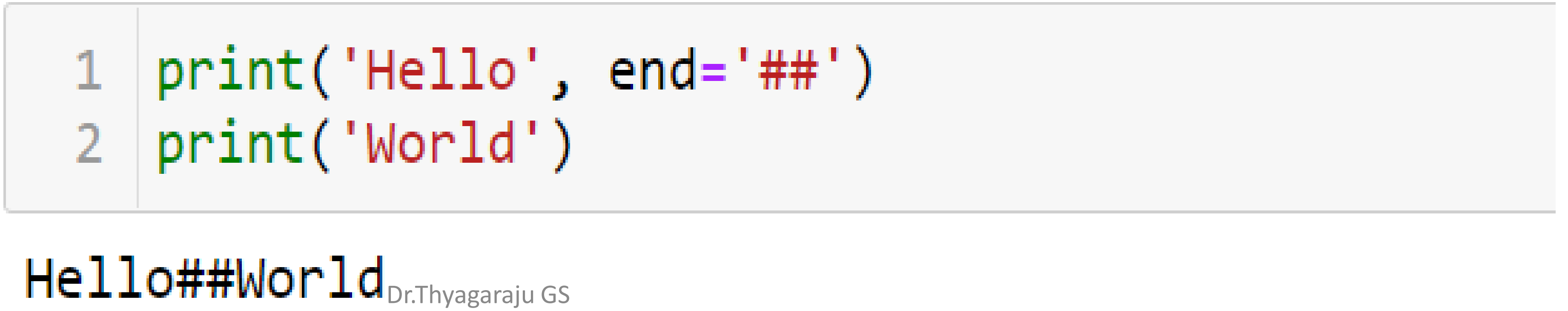
```python
print('Hello', end=' ')
print('World')
```

```
Hello World
```



```python
print('Hello')
print('World')
```

```
Hello
World
```



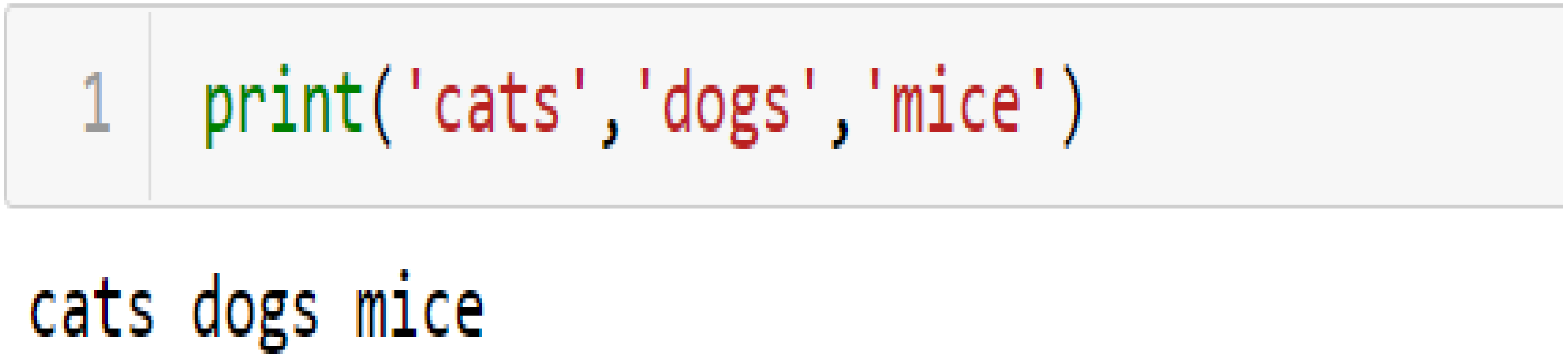
```python
print('Hello', end='##')
print('World')
```

```
Hello##World
```

Keyword Arguments and print()

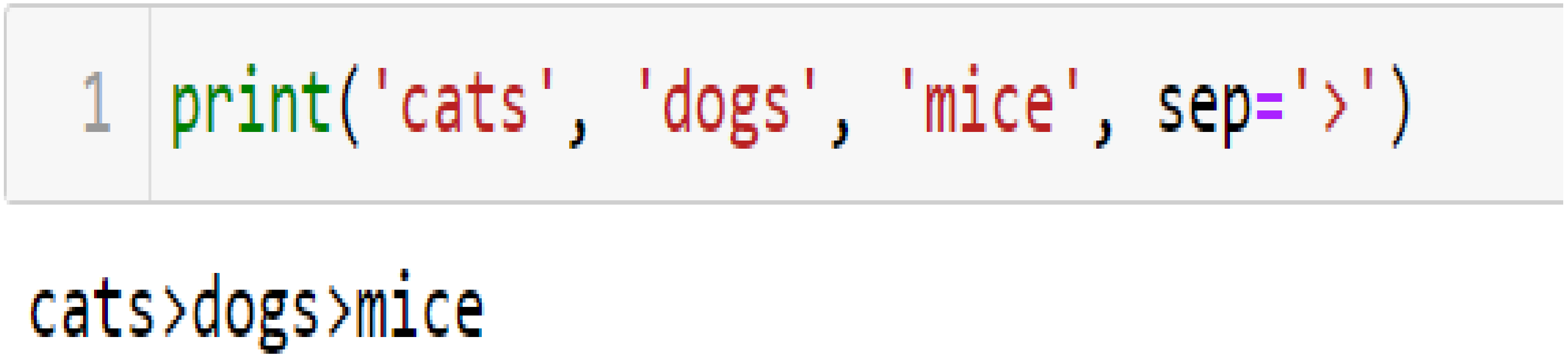
Keyword arguments are identified by the keyword put before them in the function call. Keyword arguments are often used for optional parameters.

Examples : ***end*** and ***sep***.



```python
print('cats','dogs','mice')
```

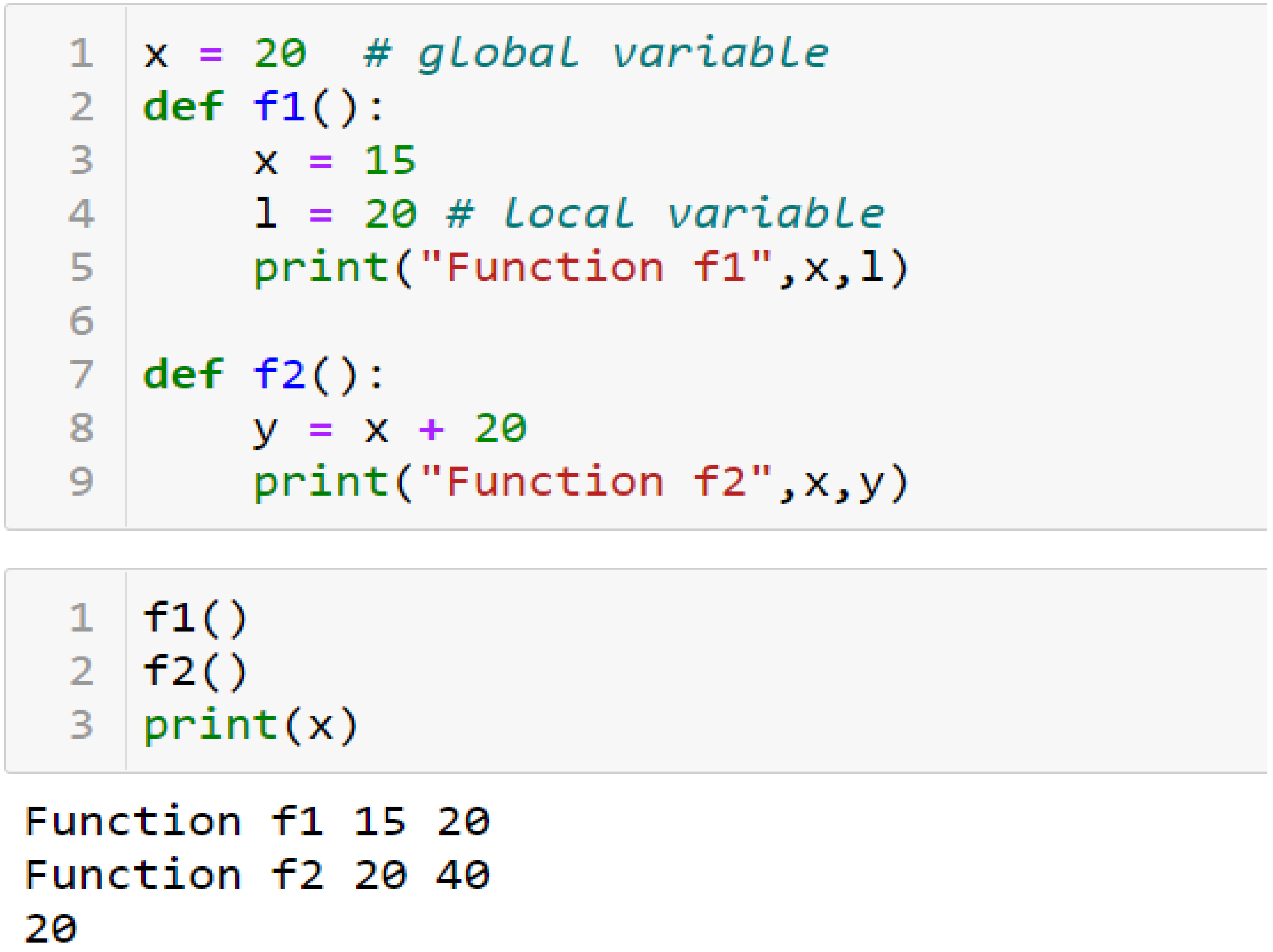
```
cats dogs mice
```



```python
print('cats', 'dogs', 'mice', sep='>')
```

```
cats>dogs>mice
```

Local and Global Scope

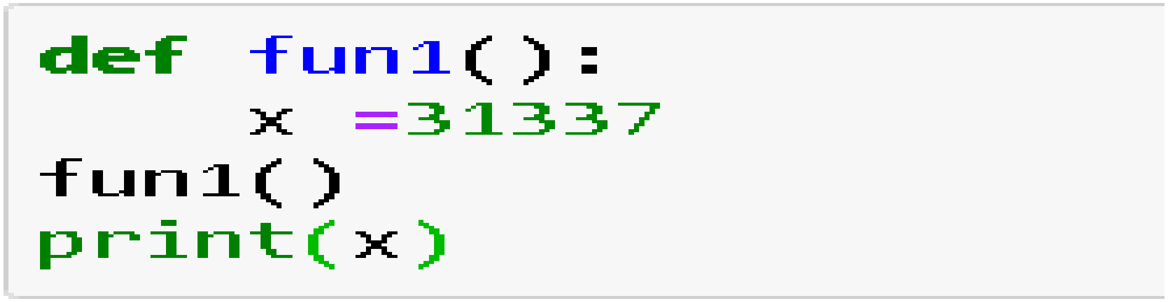


```
x = 20  # global variable
def f1():
    x = 15
    l = 20 # local variable
    print("Function f1",x,l)

def f2():
    y = x + 20
    print("Function f2",x,y)
```

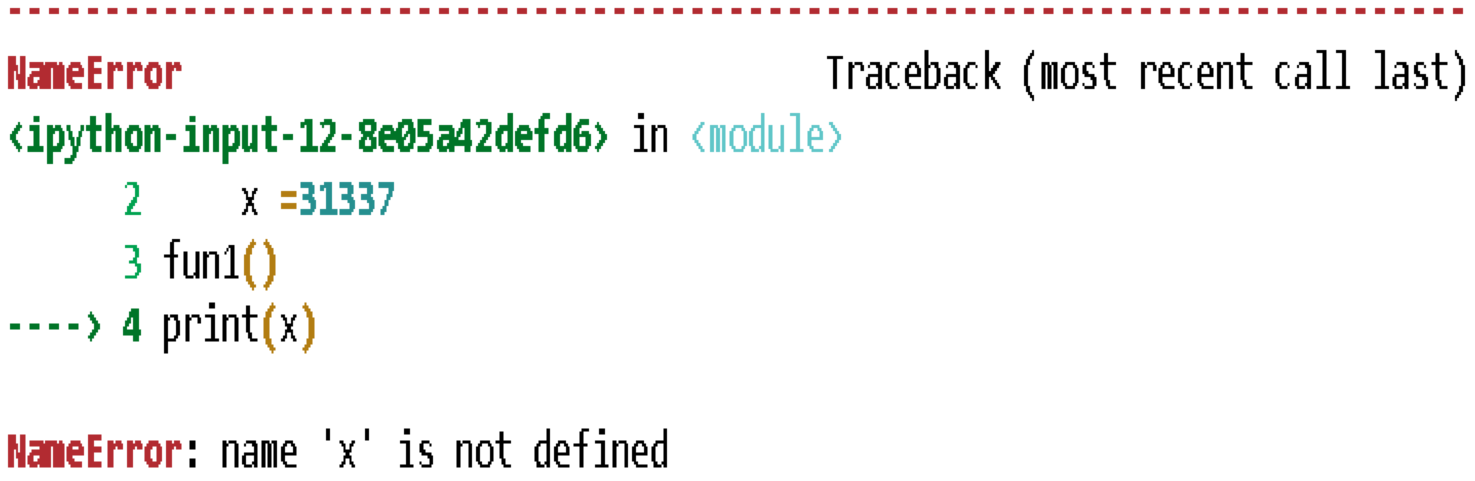
```
f1()
f2()
print(x)
```

```
Function f1 15 20
Function f2 20 40
20
```

Local variable cannot be used in the global scope



```
def fun1():
    x =31337
fun1()
print(x)
```



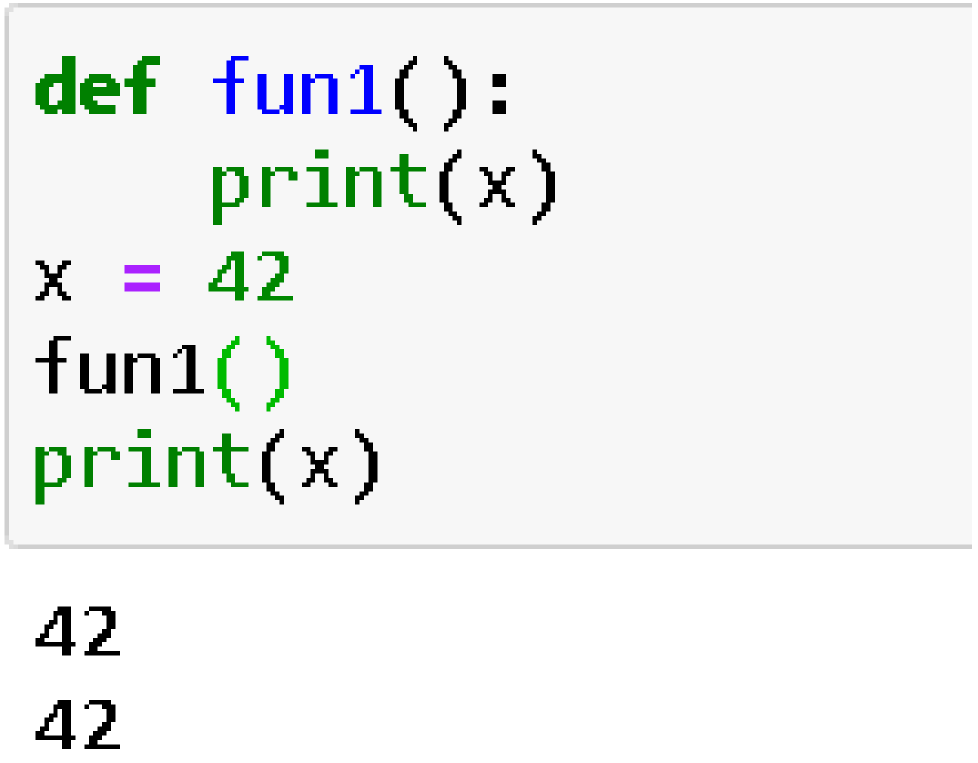
```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-12-8e05a42defd6> in <module>
      2     x =31337
      3 fun1()
----> 4 print(x)

NameError: name 'x' is not defined
```

Local Scopes Cannot Use Variables in Other Local Scopes

```
def fun1():
    x =10
    fun2()
    print(x)
def fun2():
    y = 21
    x = 0
fun1()
```

```
10
```

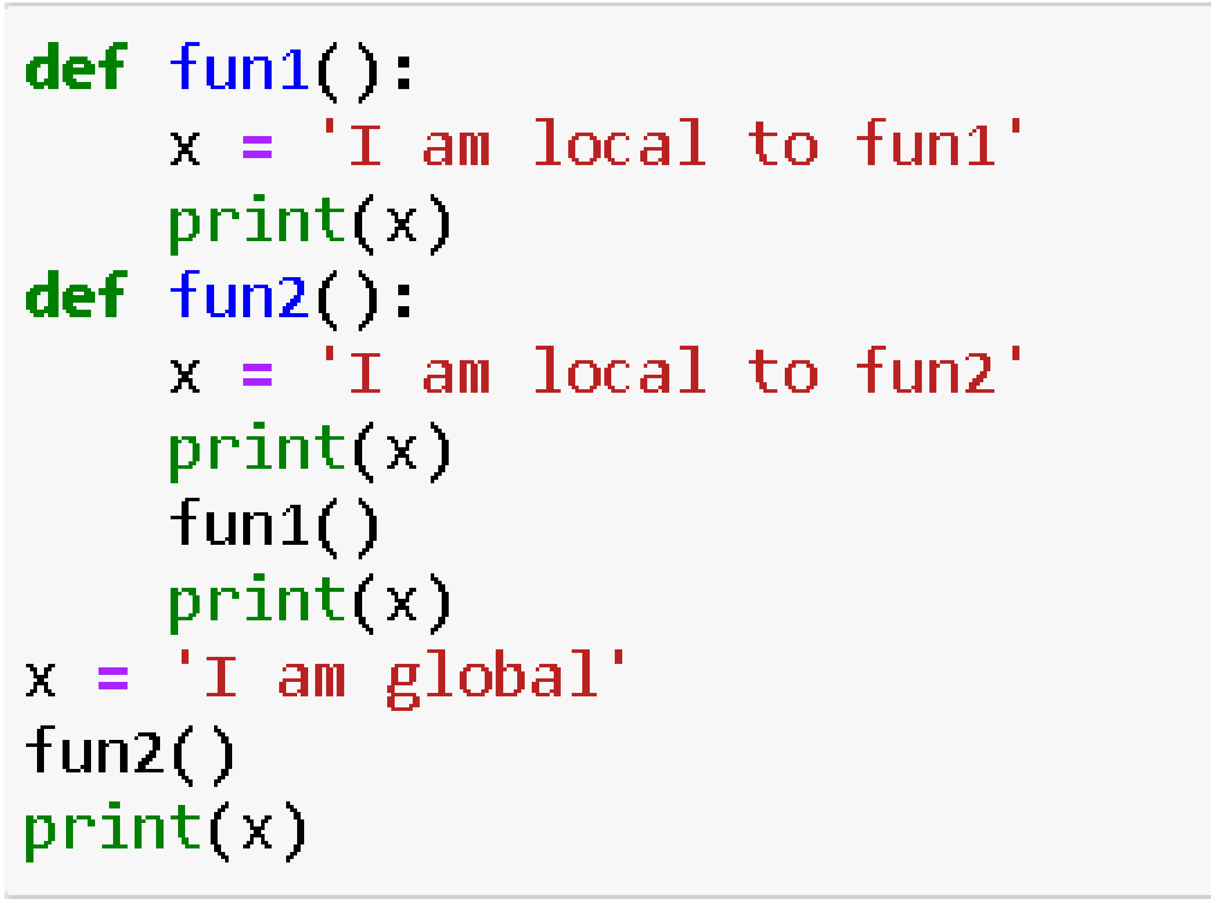
Global Variable Can be read from a local scope :



```
def fun1():
    print(x)
x = 42
fun1()
print(x)
```

```
42
42
```

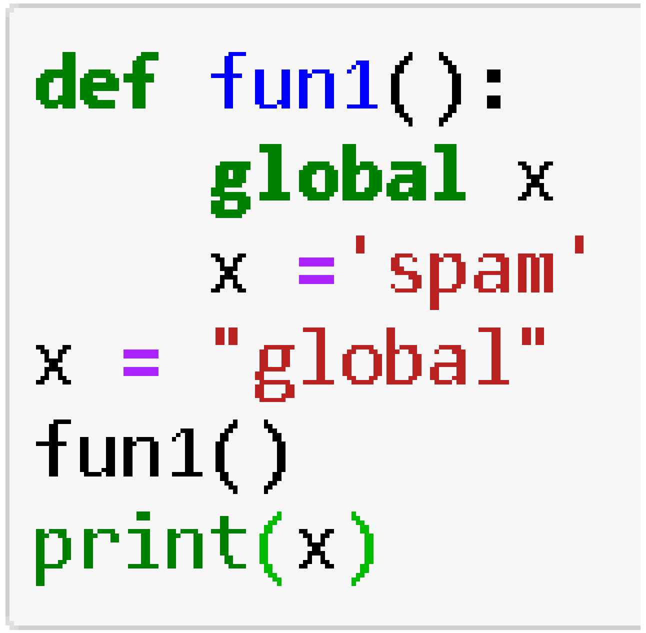
Local and Global Variables with the same Name :



```
def fun1():
    x = 'I am local to fun1'
    print(x)
def fun2():
    x = 'I am local to fun2'
    print(x)
    fun1()
    print(x)
x = 'I am global'
fun2()
print(x)
```

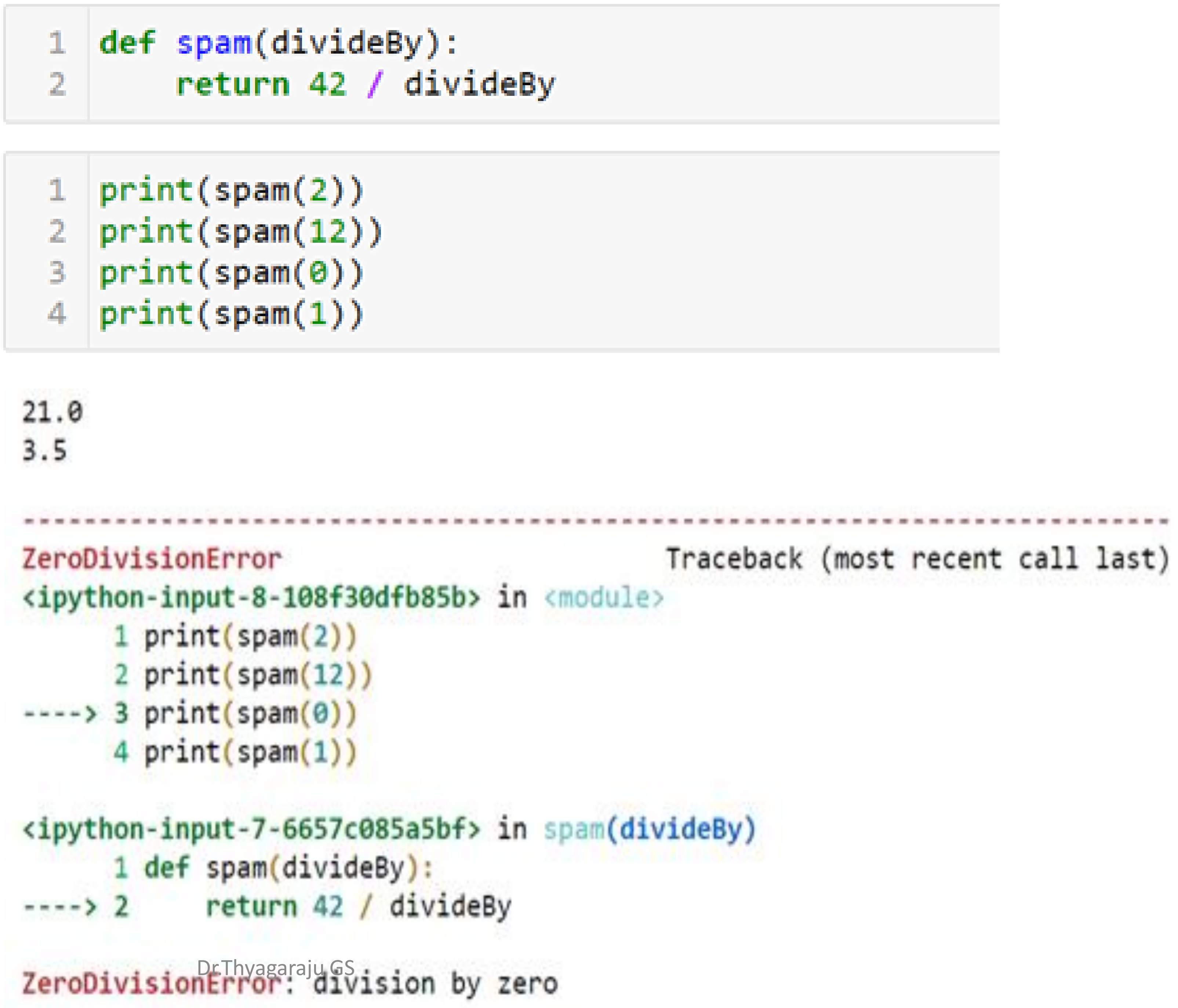
```
I am local to fun2
I am local to fun1
I am local to fun2
I am global
```

Global Statement:



```
def fun1():
    global x
    x ='spam'
x = "global"
fun1()
print(x)
```

Exceptional Handling

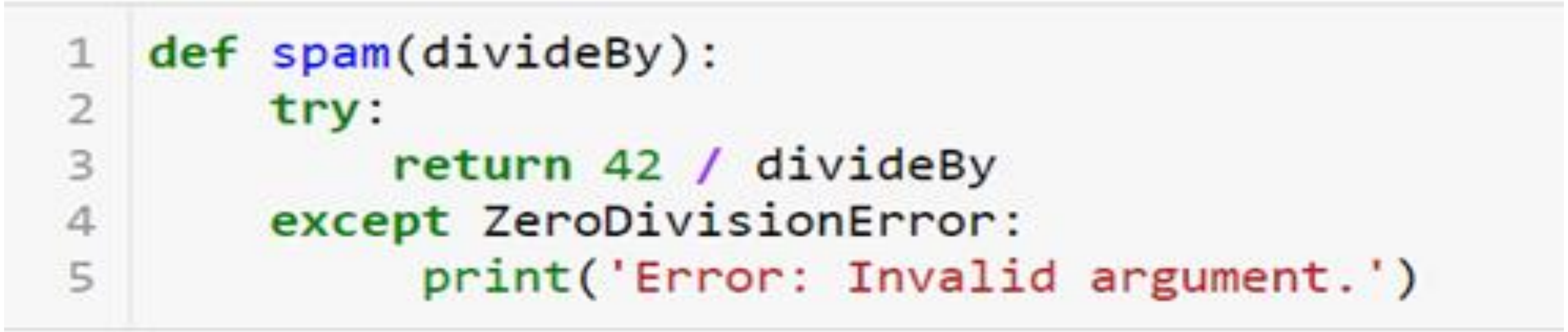


```
def spam(divideBy):
    return 42 / divideBy
```

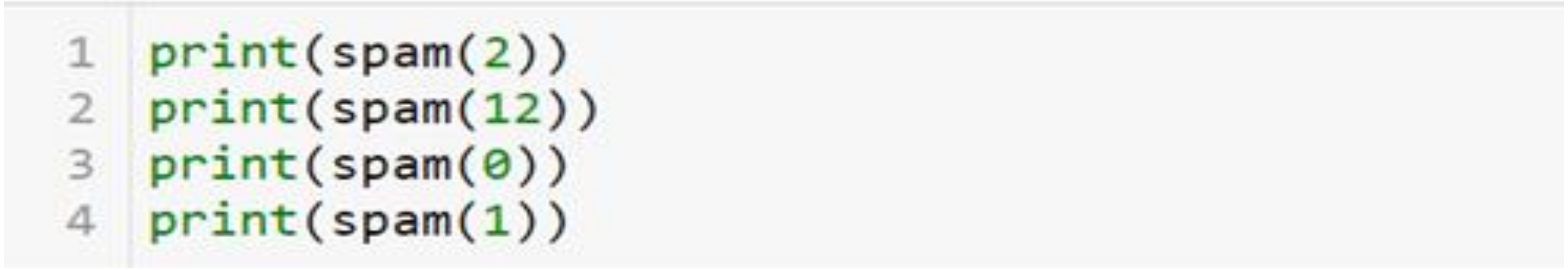
```
print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

```
21.0
3.5

---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-8-108f30dfb85b> in <module>
      1 print(spam(2))
      2 print(spam(12))
----> 3 print(spam(0))
      4 print(spam(1))

<ipython-input-7-6657c085a5bf> in spam(divideBy)
      1 def spam(divideBy):
----> 2     return 42 / divideBy

ZeroDivisionError: division by zero
```

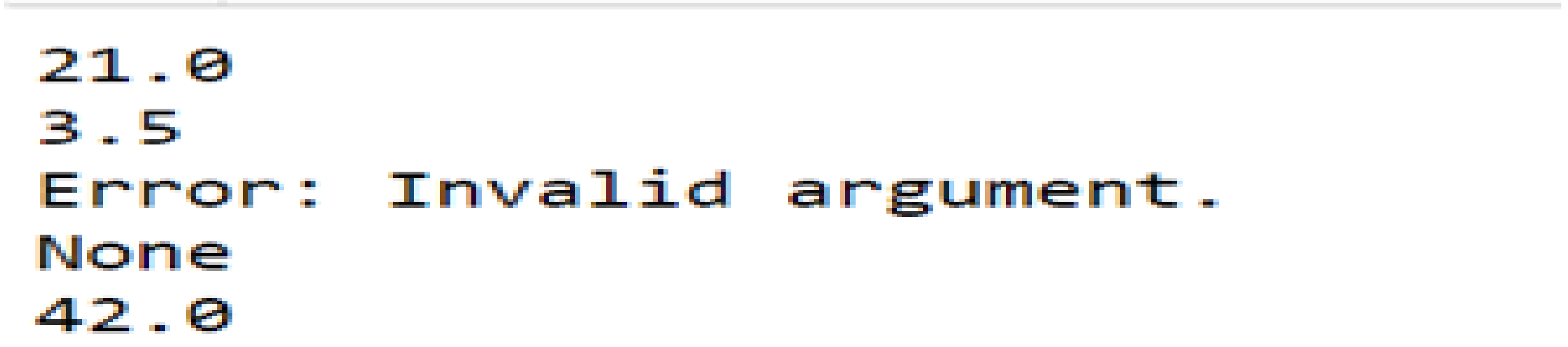
Try and Except :



```python
def spam(divideBy):
    try:
        return 42 / divideBy
    except ZeroDivisionError:
        print('Error: Invalid argument.')
```

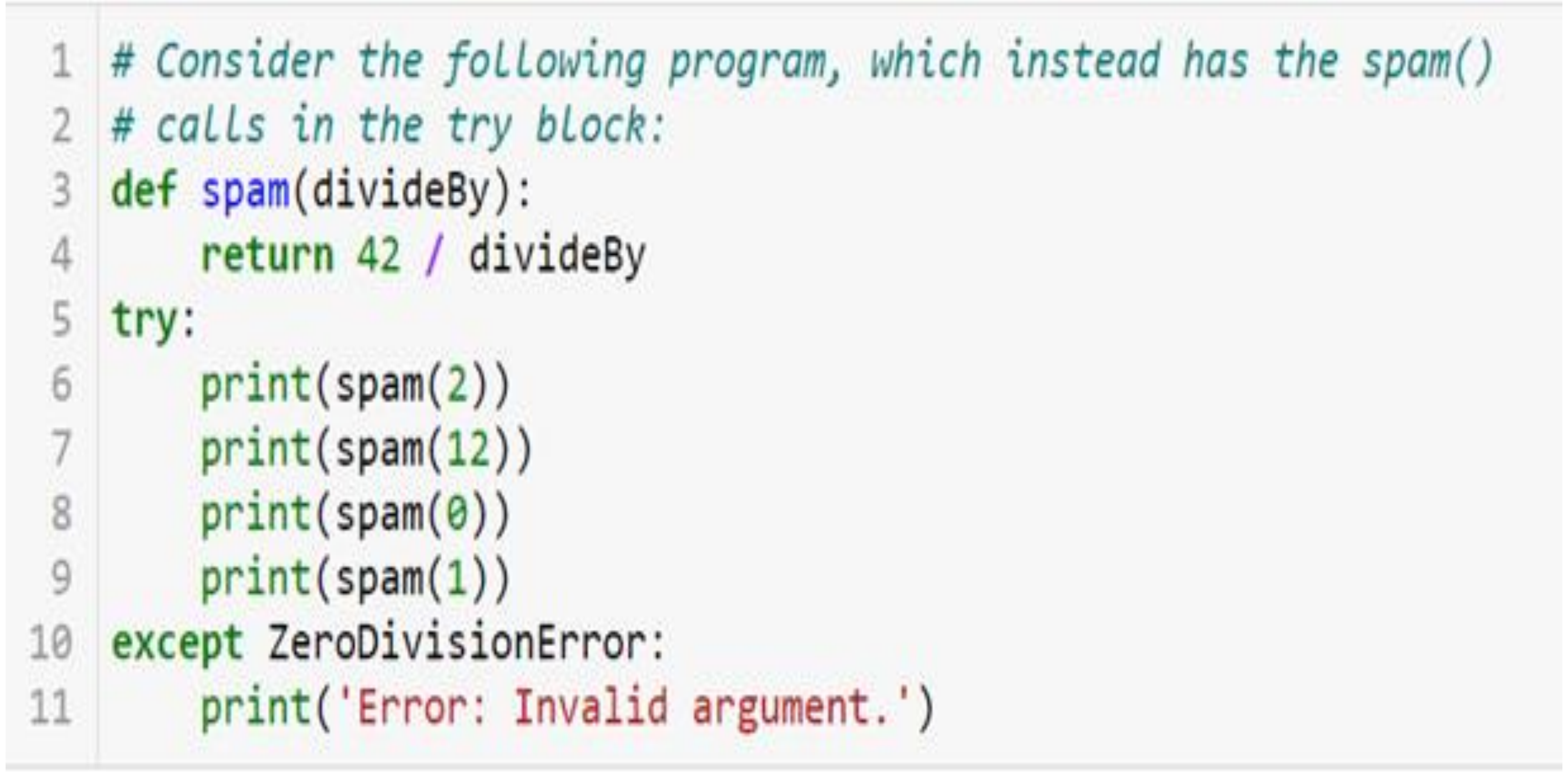


```python
print(spam(2))
print(spam(12))
print(spam(0))
print(spam(1))
```

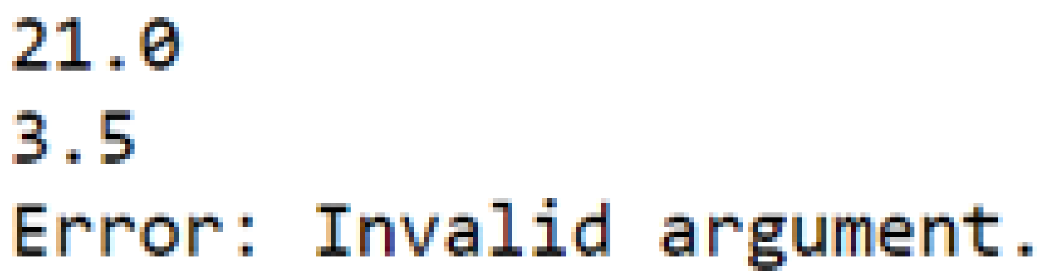


```
21.0
3.5
Error: Invalid argument.
None
42.0
```

Try and Except :

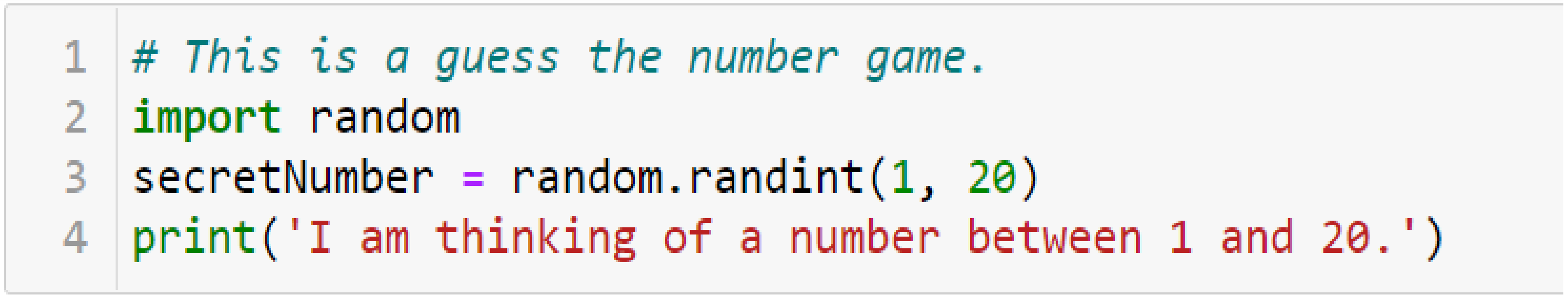


```python
# Consider the following program, which instead has the spam()
# calls in the try block:
def spam(divideBy):
    return 42 / divideBy
try:
    print(spam(2))
    print(spam(12))
    print(spam(0))
    print(spam(1))
except ZeroDivisionError:
    print('Error: Invalid argument.')
```



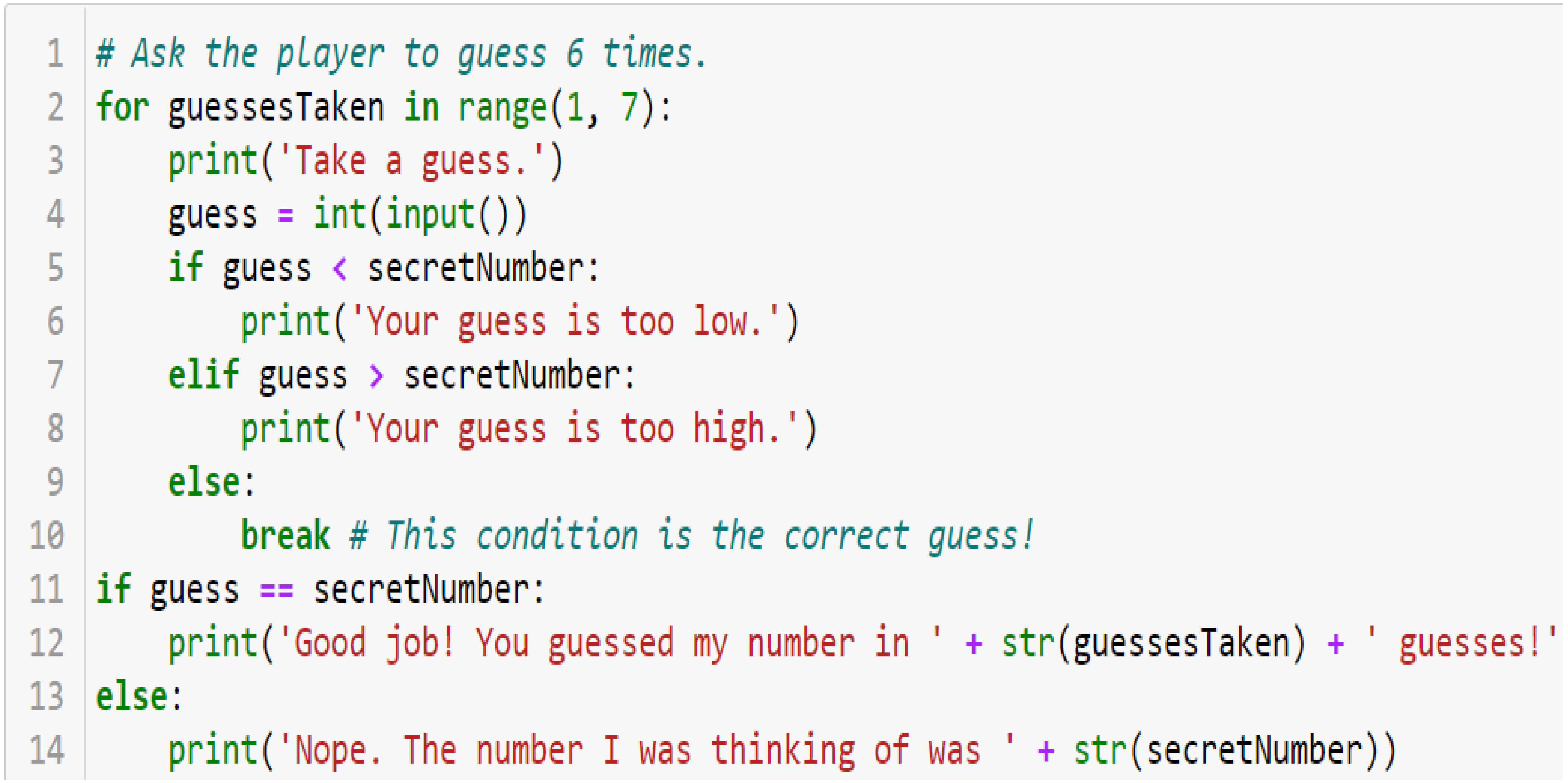
```
21.0
3.5
Error: Invalid argument.
```

A Sample Program: Guess The number



```python
# This is a guess the number game.
import random
secretNumber = random.randint(1, 20)
print('I am thinking of a number between 1 and 20.')
```

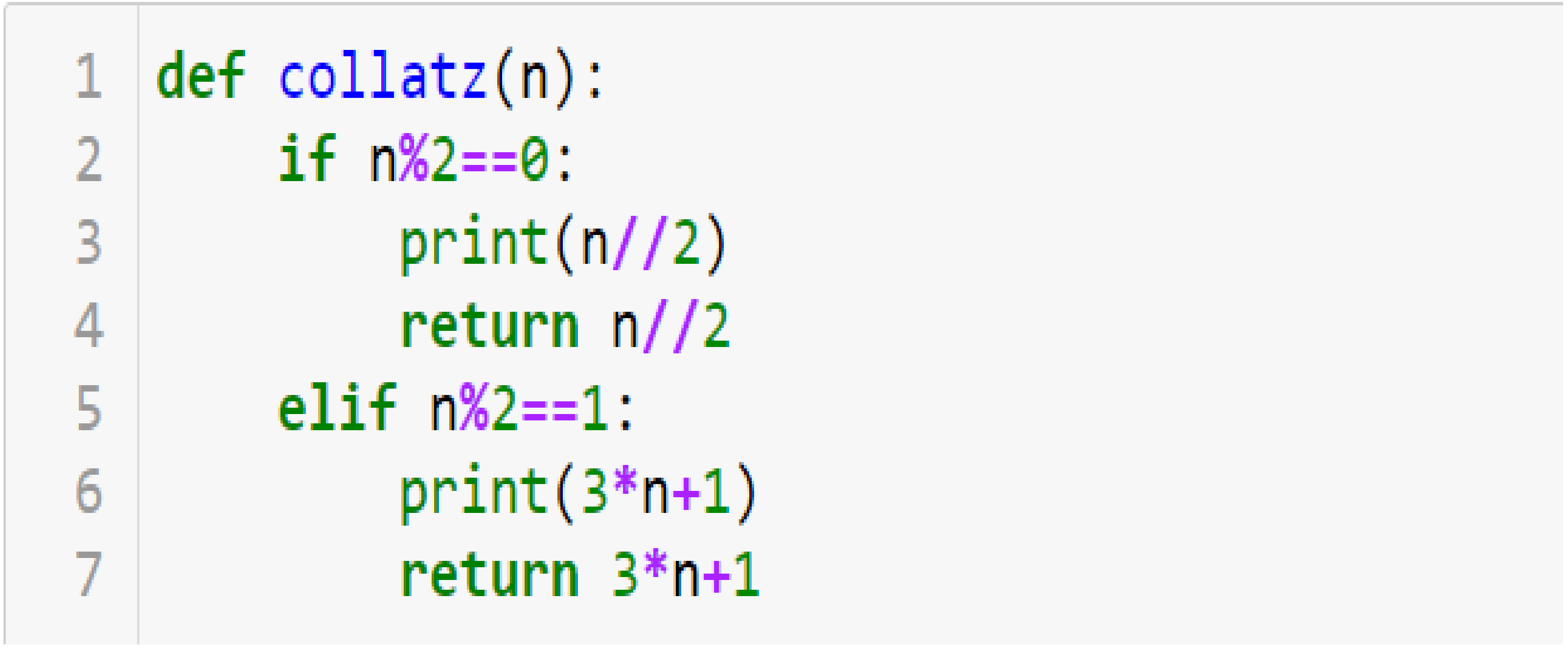
```
I am thinking of a number between 1 and 20.
```



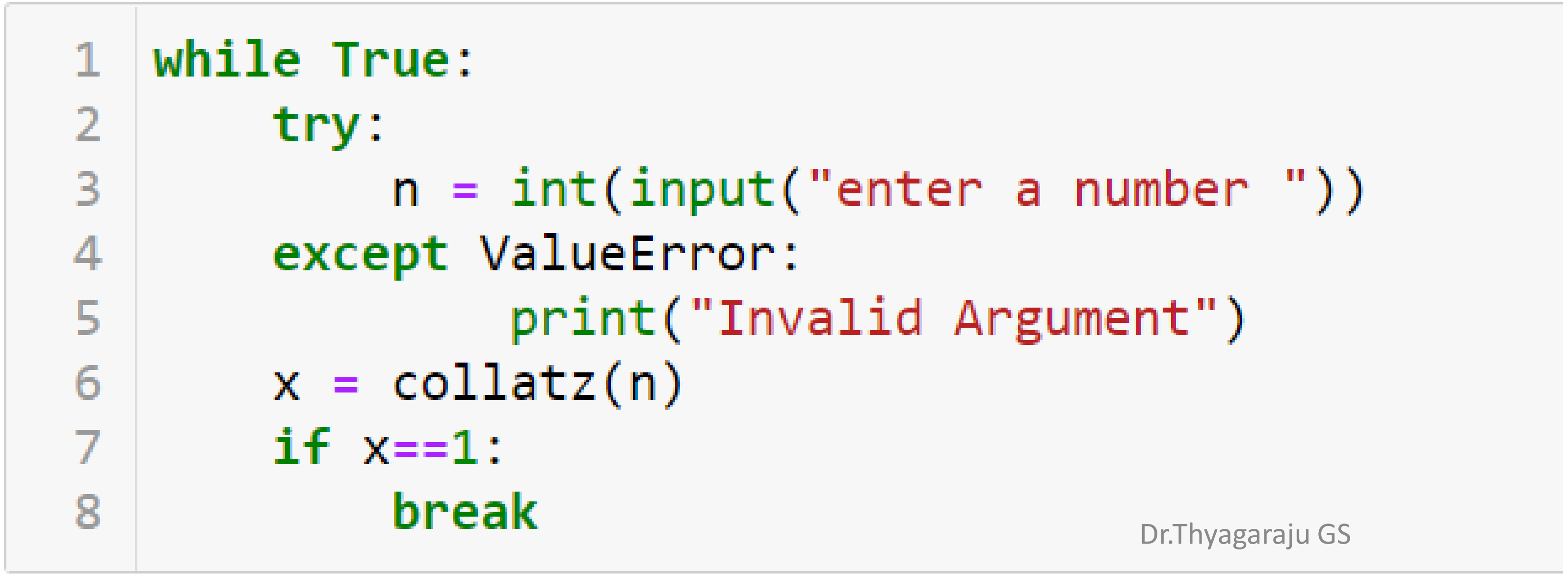
```python
# Ask the player to guess 6 times.
for guessesTaken in range(1, 7):
    print('Take a guess.')
    guess = int(input())
    if guess < secretNumber:
        print('Your guess is too low.')
    elif guess > secretNumber:
        print('Your guess is too high.')
    else:
        break # This condition is the correct guess!
if guess == secretNumber:
    print('Good job! You guessed my number in ' + str(guessesTaken) + ' guesses!'
else:
    print('Nope. The number I was thinking of was ' + str(secretNumber))
```

```
Take a guess.
8
Your guess is too low.
Take a guess.
17
Your guess is too high.
Take a guess.
13
Your guess is too low.
Take a guess.
15
Your guess is too high.
Take a guess.
14
Good job! You guessed my number in 5 guesses!
```

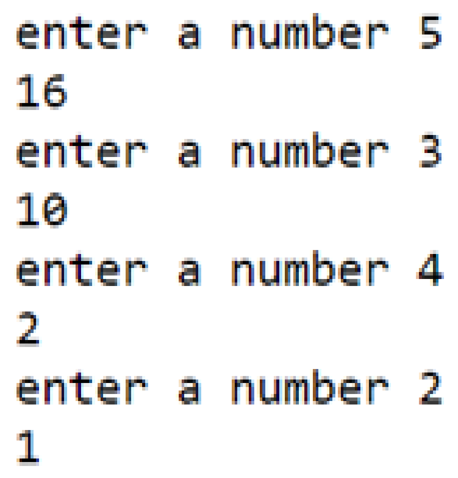
Simple Project



```python
def collatz(n):
    if n%2==0:
        print(n//2)
        return n//2
    elif n%2==1:
        print(3*n+1)
        return 3*n+1
```



```python
while True:
    try:
        n = int(input("enter a number "))
    except ValueError:
            print("Invalid Argument")
    x = collatz(n)
    if x==1:
        break
```



```
enter a number 5
16
enter a number 3
10
enter a number 4
2
enter a number 2
1
```